TÉCNICO LISBOA

Highly Dependable Systems

Lecture 1

Course introduction

Lecturers: Miguel Matos and Rodrigo Miragaia Rodrigues

Instructors - Lectures

- Prof. Miguel Matos
  - Office hours: Friday afternoon from 19pm (Tagus/Discord)
- Prof. Rodrigo Miragaia Rodrigues
  - Office hours: Monday morning from 8am (INESC-ID)

Instructors - Labs

- Guilherme Jardim
- Rafael Gonçalves
- Sidnei Teixeira
- Diogo Pacheco

Contacts

- For administrative / exam questions please contact both:
  - miguel.marques.matos@tecnico.ulisboa.pt
  - rodrigo.miragaia.rodrigues@tecnico.ulisboa.pt
- For lab / project questions, Discord and TA’s Office Hours

Syllabus

- Dependability fundamentals
- Security and crypto refresher
- Blockchain fundamentals
  - Byzantine fault-tolerant primitives and consensus
  - Other consensus (proof of work, proof of stake, etc.)
  - Blockchain systems
  - Applications: cryptocurrencies, smart contracts, etc.
- Trusted computing

Bibliography

- First half of the course follows closely the following book:
  - Introduction to Reliable and Secure Distributed Programming, 2nd Edition. C. Cachin, R. Guerraoui, L. Rodrigues 2011. Springer - ISBN: 978-3-642-15259-7
- We will post other materials online

Evaluation methods

- Exam (50%)
  - Minimum grade: 8
- Project (40%)
  - Group of 3 students, minimum grade: 8
- Paper presentation and reviewing (10%)
  - Same group as the project

- The first 2 components are mandatory
  - Failing any them → failing this course

Evaluation methods – project

- 2 stages
- Evaluation based on demo, report, code review and discussion
- Grading:
- if grade of 2nd stage is higher than the grade of 1st stage then
  - project grade = grade of the 2nd stage
- else
  - project grade = 50% 1st stage + 50% 2nd stage

Evaluation methods – paper presentation

- Recent papers from top conferences
- Each group presents one paper in one of the last two lectures
  - Technical content
  - Key ideas
  - Positive aspects
  - Negative aspects
  - Reviewing the paper – would you accept this paper if you were on the program committee? Why or why not?
- Questions from the audience
  - Not only clarifications – we expect lots of insightful criticism

Lab info

- Lab registration open through fénix
- If needed, ok for project groups to span different labs or different campi
  - Across campi, presentations and discussions must be in Tagus
- First lab class will be intro to crypto API in Java
  - Due to Carnival break → intro lab in 2nd week of classes
- Subsequent classes will focus on project guidance

Questions?

What is dependability?

- A system is dependable if it is able to work (i.e., deliver its service) in a way that is justifiably trusted, namely avoiding service failures that are either more frequent or severe than acceptable.
- Concepts of dependability and trust are intertwined: system A trusts system B if it accepts placing a dependence on system B.

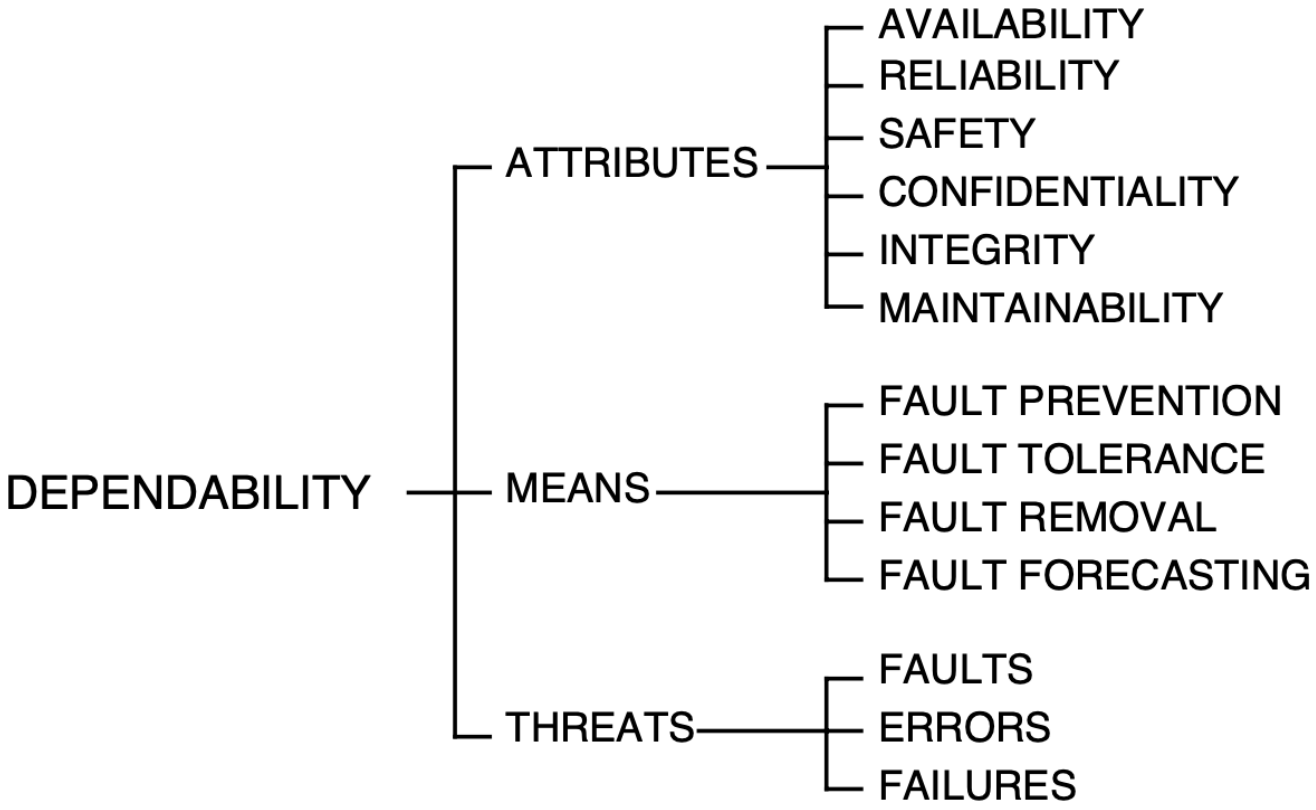
Early history

- First generation of electronic computers (late 1940s to mid 1950s) used rather unreliable components
- Hardware and software redundancy methods were developed (including foundational work from von Neumann, Moore, Shannon in the mid-50s)

A tale of two communities

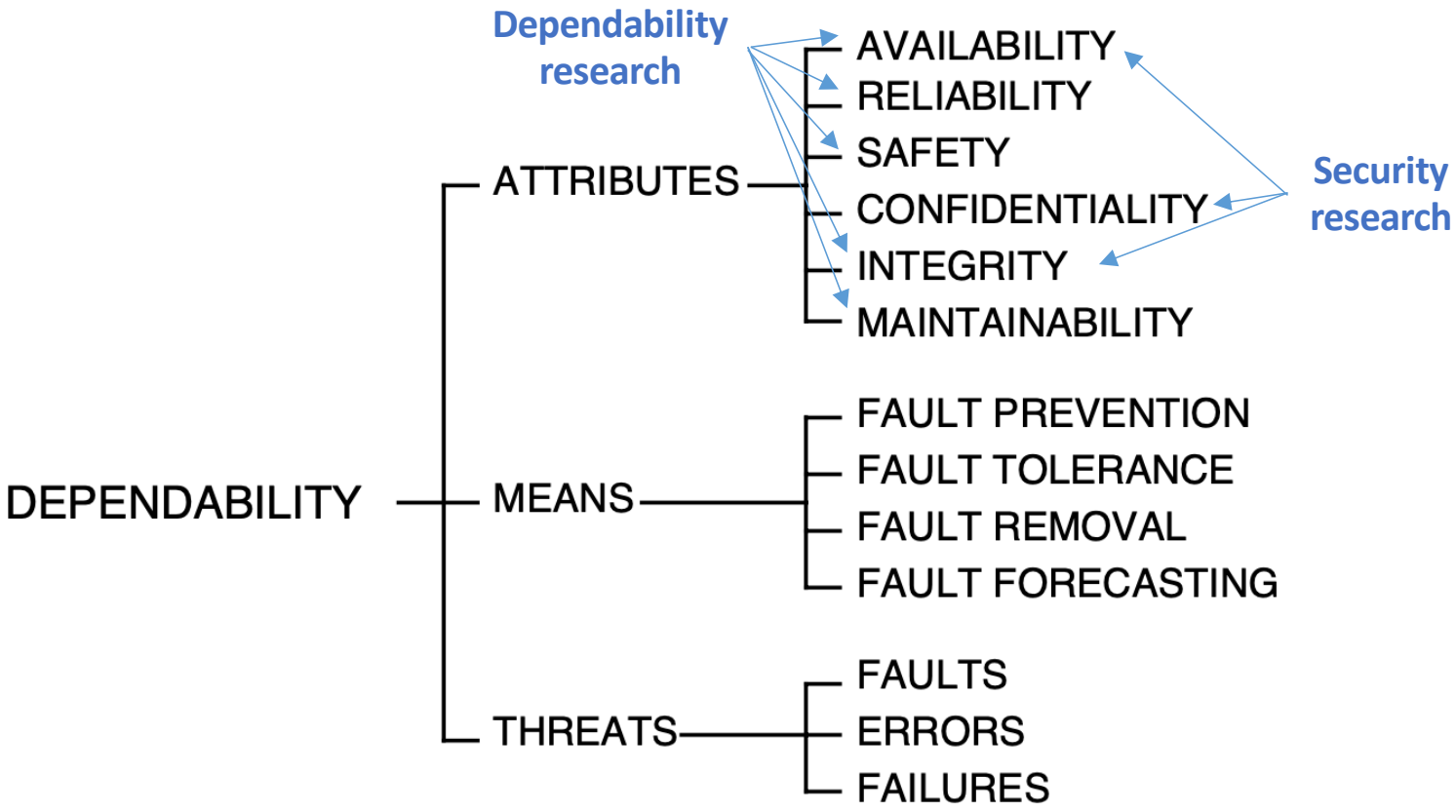
- Dependable systems community evolved from this work, initially focusing on non-malicious faults (machines or components suddenly halting or malfunctioning)
- In parallel, security research community looked at intrusions in computer systems
- Dependability realizes the convergence between the two:
  - Non-malicious faults are only one of the parts of building dependable systems
  - Security research focused on confidentiality, but needed to pay more attention to availability and integrity

Main dependability concepts



- For more details, read: Avizienis, Laprie, Randell. “Fundamental Concepts of Dependability” – available in course fénix page

Main dependability concepts



- For more details, read: Avizienis, Laprie, Randell. “Fundamental Concepts of Dependability” – available in course fénix page

Dependability and security attributes

- Availability
  - *Readiness* for correct service
  - Instantaneous
  - Expressed as a percentage representing the ratio of uptime to total time
- Reliability
  - *Continuity* of correct service
  - Expectation of the predictable remaining uptime, measured using metrics such as Mean Time Between Failures (MTBF), or Mean Time to Fail (MTTF)

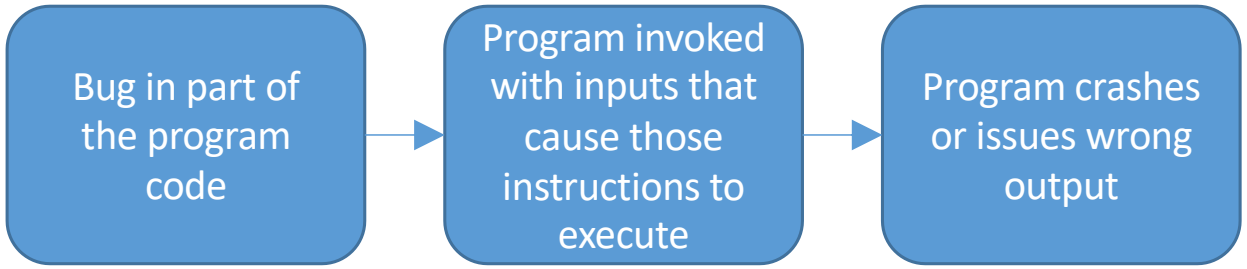
Dependability and security attributes

- Safety
  - Absence of catastrophic consequences on the user(s) and the environment
- Confidentiality
  - Absence of unauthorized disclosure of information
- Integrity
  - Absence of improper system alterations
- Maintainability
  - Ability to undergo modifications and repairs

Threats: fault, error, failure

- Faults
  - A fault is the hypothesized cause of an error
- Errors
  - Part of total state of the system that may lead to subsequent service failure
    - Active
    - Latent
- Failure
  - Event that occurs when the delivered service deviates from correct service
  - Time between failure and service restoration is called an outage

Fault, error, and failure by example



Means to attain dependability

- Fault prevention / avoidance
  - means to prevent the occurrence or introduction of faults
- Fault removal
  - means to reduce the number and severity of faults (testing, verification, fault injection)
- Fault forecasting / prediction
  - identify components that are likely to become a source of faults and when they fail
- Fault tolerance
  - means to avoid service failures in the presence of faults

Many concepts are widely used today:

Source: https://soufianebouchaa.medium.com/sla-and-slo-fundamentals-and-how-to-calculate-sla-7f3e75831f79

Soufiane Bouchaara Follow
Mar 5, 2022 · 6 min read · Listen

Save

SLA and SLO fundamentals and how to calculate SLA

SLA aka Service-Level Agreement is an agreement you make with your clients/users, which is a measured metric that can be time-based or aggregate-based.

SLA time-based

We can calculate the **tolerable duration of downtime** to reach a given number of nines of availability, using the following formula:

$$\text{availability} = \frac{\text{uptime}}{(\text{uptime} + \text{downtime})}$$

For example, a web application with an availability of **99.95%** can be down for up to **4.38 hours max in a year**.

The following table explains the maximum duration of tolerated downtime per year/month/week/day/hour.

	per year	per month	per week	per day	per hour
90%	36.5 days	3 days	16.8 hours	2.4 hours	6 minutes
95%	18.25 days	1.5 days	8.4 hours	1.2 hours	3 minutes
99%	3.65 days	7.2 hours	1.68 hours	14.4 minutes	36 seconds
99.50%	1.83 days	3.6 hours	50.4 minutes	7.20 minutes	18 seconds
99.90%	8.76 hours	43.2 minutes	10.1 minutes	1.44 minutes	3.6 seconds
99.95%	4.38 hours	21.6 minutes	5.04 minutes	43.2 seconds	1.8 seconds
99.99%	52.6 minutes	4.32 minutes	60.5 seconds	8.64 seconds	0.36 seconds
99.999%	5.26 minutes	25.9 seconds	6.05 seconds	0.87 seconds	0.04 seconds

Important output of dependability research (we will see why in a few slides)

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound two loyal generals. With unforgeable written messages, the problem is solvable for any number of generals and possible traitors. Applications of the solutions to reliable computer systems are then discussed.

Categories and Subject Descriptors: C.2.4. [**Computer-Communication Networks**]: Distributed

Byzantine consensus (or agreement) problem

- Defined by Lamport in late 1970s / early 80s
- Treats malicious behavior as arbitrary (a.k.a. Byzantine) faults
- N processes (machines), up to f may fail in an arbitrary fashion
- Each has an input value, all must agree on common output
- For the first two decades (80s, 90s) little attention was paid to Byzantine fault tolerance (BFT)

PBFT (1999)
(Can we get good performance from BFT?)

Appears in the *Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, February 1999*

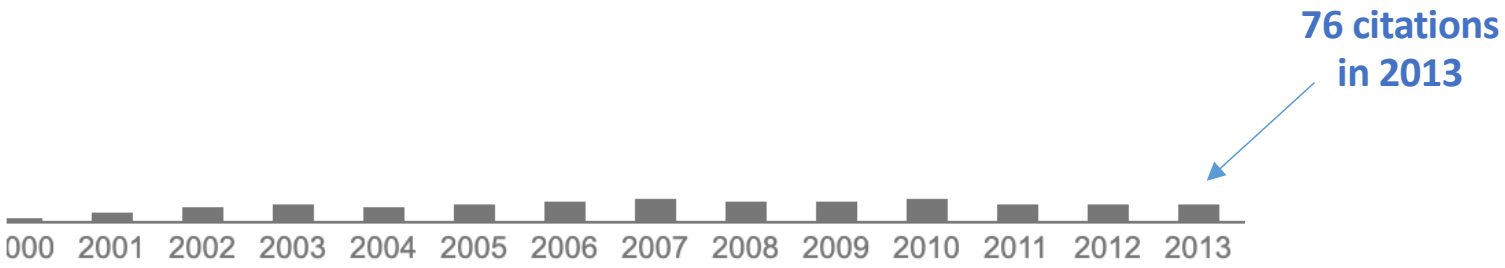
Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov
Laboratory for Computer Science,
Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139
{castro,liskov}@lcs.mit.edu

Abstract

This paper describes a new replication algorithm that is able to tolerate Byzantine faults. We believe that Byzantine-fault-tolerant algorithms will be increasingly important in the future because malicious attacks and software errors are and replication techniques that tolerate Byzantine faults (starting with [19]). However, most earlier work (e.g., [3, 24, 10]) either concerns techniques designed to demonstrate theoretical feasibility that are too inefficient to be used in practice, or assumes synchrony, i.e.,

Picked up some attention, but wore off

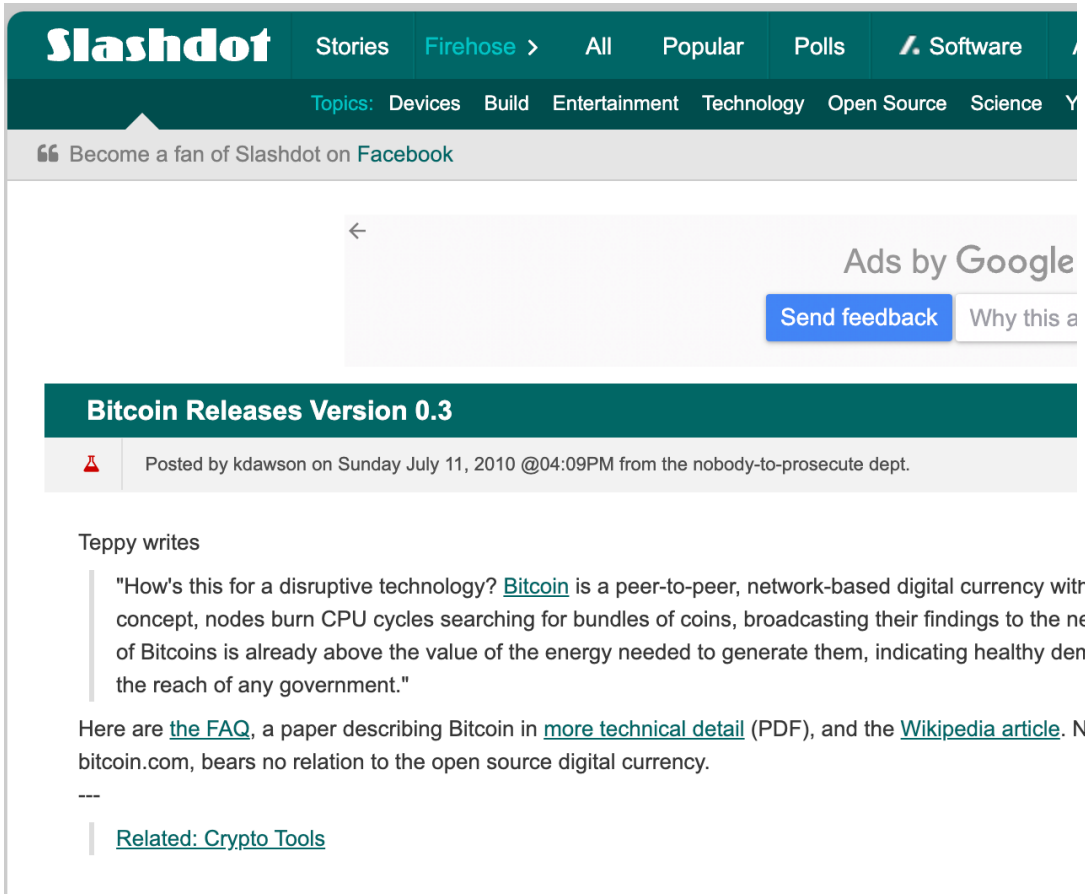


Scholar articles Practical byzantine fault tolerance
M Castro, B Liskov - OsDI, 1999
Cited by 5017 Related articles All 118 versions

Until a decade later…

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org



Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Last Updated: 19th February 2016

Who is Satoshi Nakamoto?

While we may not know who he (or she)
was the inventor of the bitcoin protocol,
List in November 2008.

ed the first version of the

The Economist

World politics | Business & finance | Economics | Science & technology | Culture

Bitcoin's creator

Craig Steven Wright claims to be Satoshi Nakamoto. Is he?

All latest updates

Evaluating his claim will involve a multi-

May 2nd 2016 | Online extra

2.2K Tweet

Craig Wright is not Satoshi Nakamoto

Craig Wright is not Satoshi Nakamoto. He wasn't Satoshi Nakamoto before or after Wired and Gizmodo suspected him to be last year, and he still isn't Satoshi Nakamoto after trying to reveal himself to be on his own blog and to The BBC, The Economist, GQ, Jon Matonis and Gavin Andresen.

There is a long and fraught history in Bitcoin of claims and counterclaims about who Satoshi is, and one would think that lessons had been learned and a high standard would be set for subsequent claims regarding Satoshi Nakamoto. The proof posted today by Wright and others does not meet any standard for identifying him as Nakamoto.

Bitcoin is a currency based on cryptography. The ownership of coins can be proven cryptographically and verified by any network participant in a process that is at the

offered by Wright can not be independently

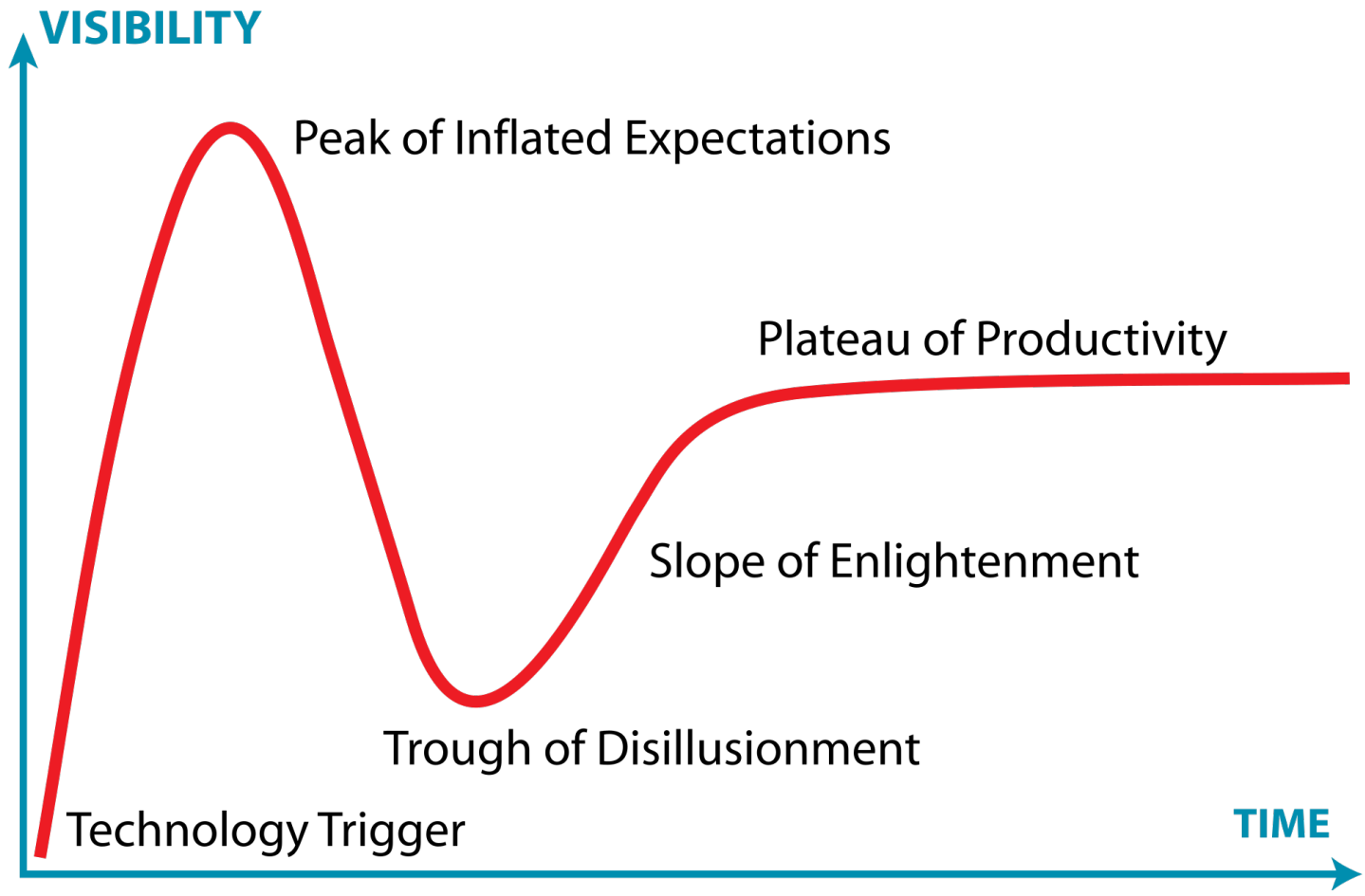
made today by

+20,282.86 (6,606.36%)↑ all time

Feb 14, 19:12 UTC · Disclaimer

1D | 5D | 1M | 6M | YTD | 1Y | 5Y | Max

Bears some resemblance?



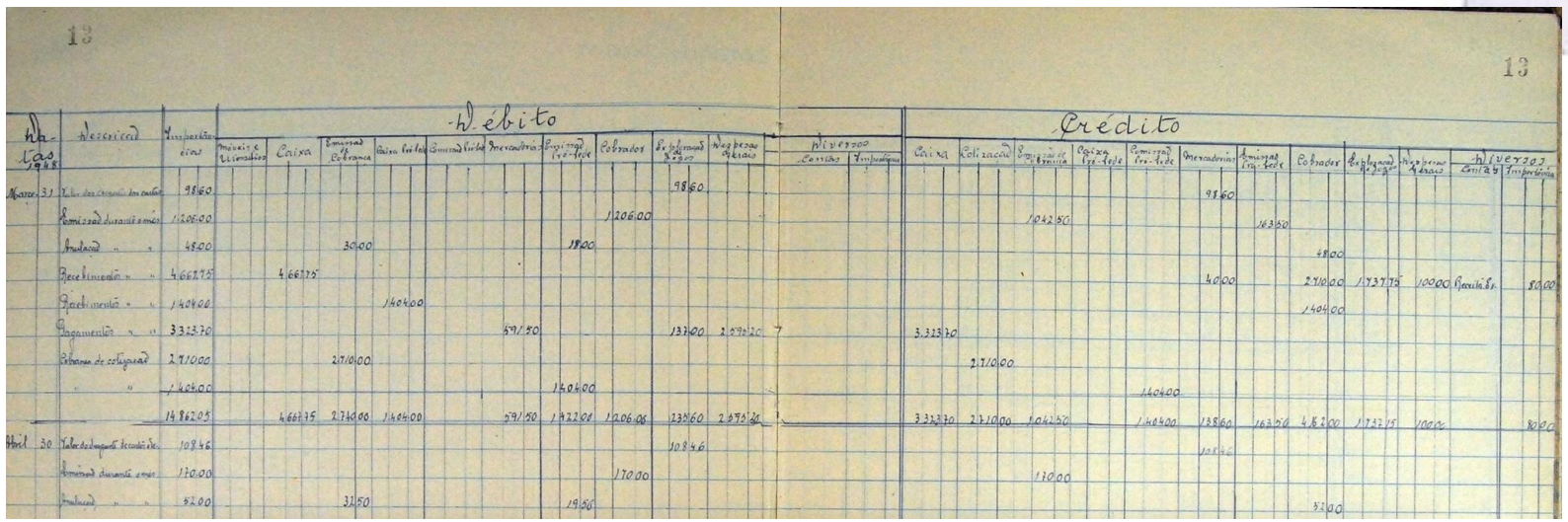
Are we living a Lehman moment or just growing pains?

This course covers the principles of **blockchains** (not just bitcoin)

- Blockchain is the technology behind bitcoin
- Crucial to understand and get the technology right
  - Thus we should treat the subject systematically, based on the principles of dependability

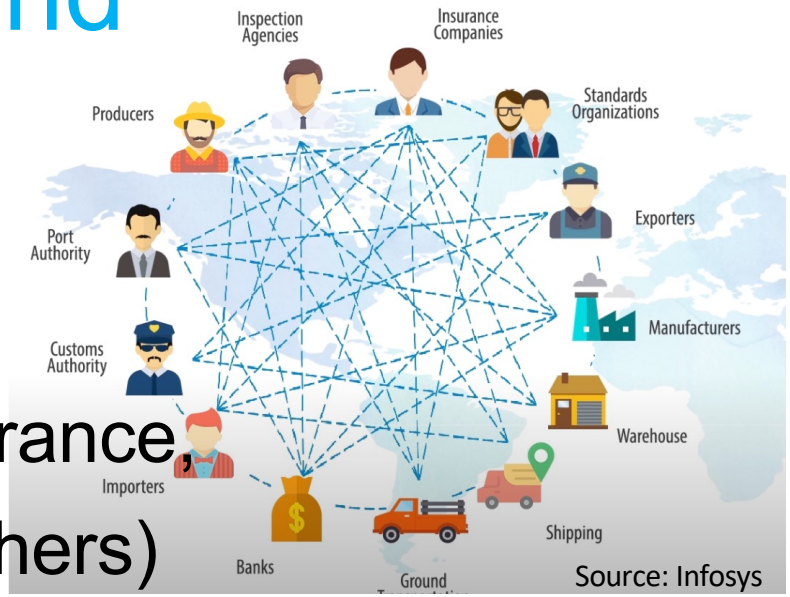
So what is a blockchain then?

- Abstraction of a ledger
  - Append-only sequence of events
  - Tamper-proof and agreed upon by different stakeholders



- Implemented in a distributed way, without requiring centralized trust

Does it have any use beyond cryptocurrencies?

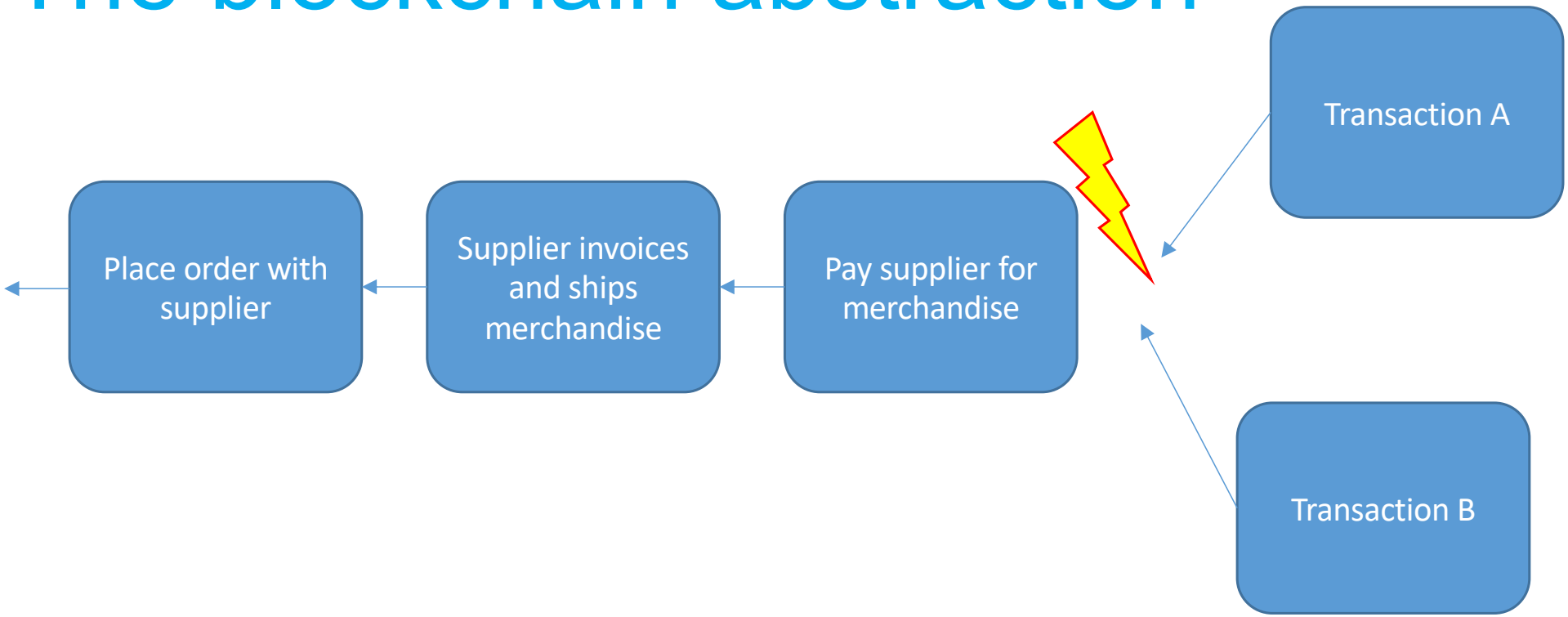


- Example: agriculture supply chain
- Stakeholders: producer, inspection, insurance, logistics, importer, consumer (among others)
- Difficult to trace as goods moves among these
- What if importer wants to know ingredients, place of origin, or ethical / sustainability guarantees of imported produce?
  - Nowadays, even more important due to food/drug safety concerns

The blockchain abstraction

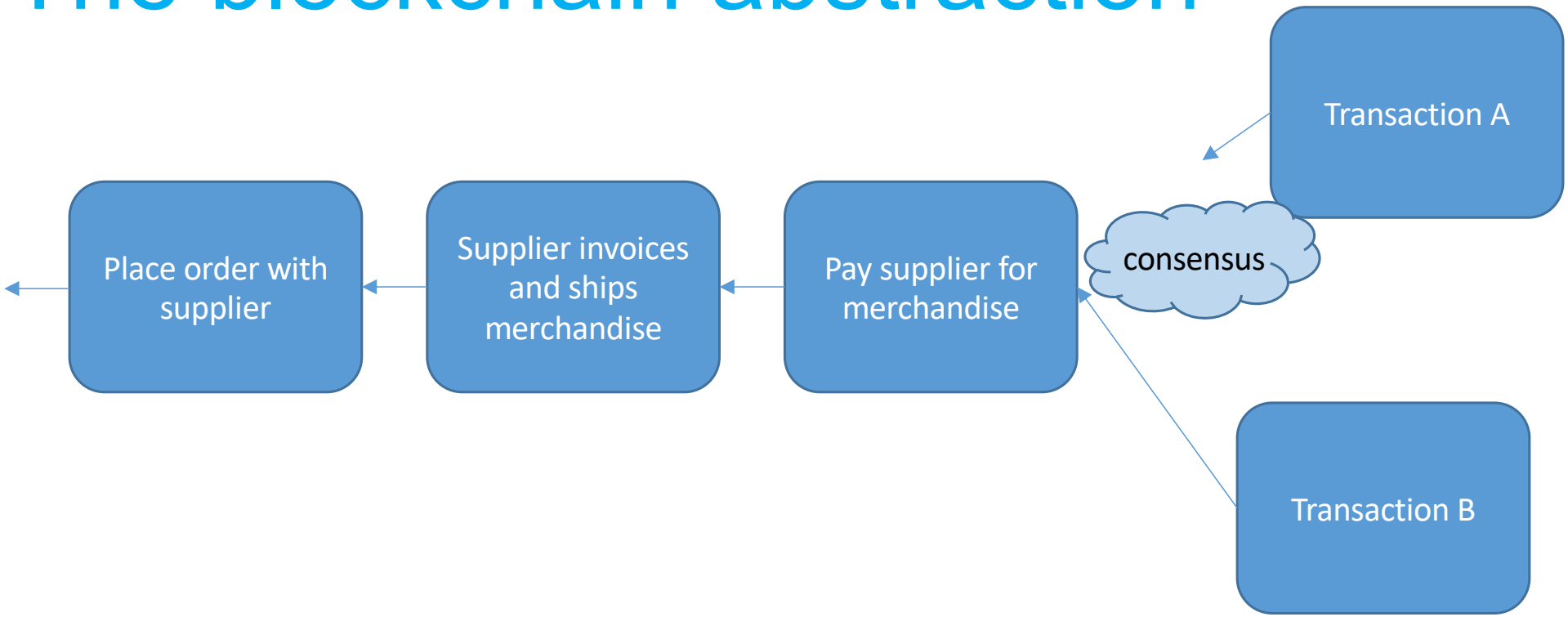
- Each newly inserted block points to its predecessor

The blockchain abstraction



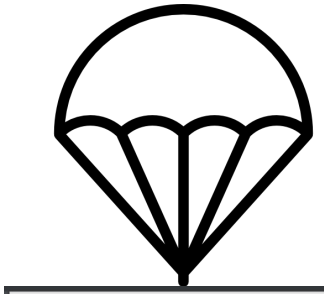
- How to resolve contention for appending a new operation?

The blockchain abstraction



- Run consensus on the next block.
- How to have multiple participants trust the output of consensus?

BFT consensus to the rescue



Appears in the Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, February 1999

Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov
Laboratory for Computer Science,
Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139
{castro,liskov}@lcs.mit.edu

Abstract

This paper describes a new replication algorithm that is able to tolerate Byzantine faults. We believe that Byzantine-fault-tolerant algorithms will be increasingly important in the future because faulty computing is increasingly expensive as increasingly common and can cause faulty nodes to exhibit arbitrary behavior. Whereas previous algorithms assumed a synchronous system or were too slow to be used in practice, the algorithm described in this paper is practical: it works in asynchronous environments like the Internet and incorporates several important optimizations that improve the response time of previous algorithms by more than an order of magnitude. We implemented a Byzantine-fault-tolerant NFS service using our algorithm and measured its performance. The results show that our service is only 3% slower than a standard unreplicated NFS.

1 Introduction

Malicious attacks and software errors are increasingly common. The growing reliance of industry and government on online information services makes malicious attacks more attractive and makes the consequences of successful attacks more serious. In addition, the number of software errors is increasing due to the growth in size and complexity of software. Even malicious attacks and software errors can cause faulty nodes to exhibit Byzantine (i.e., arbitrary) behavior. Byzantine-fault-tolerant algorithms are increasingly important.

This paper presents a new, *practical* algorithm for state-machine replication [7, 34] that tolerates Byzantine faults. The algorithm offers both liveness and safety provided at most $\lfloor (m-1)/3 \rfloor$ out of a total of m replicas are simultaneously faulty. This means that clients eventually receive replies to their requests and those replies are correct according to linearizability [14, 4]. The algorithm works in asynchronous systems like the Internet and it incorporates important optimizations that enable it to perform efficiently.

There is a significant body of work on agreement and replication techniques that tolerate Byzantine faults (starting with [18]). However, most earlier work (e.g., [3, 24, 10]) either concerns techniques designed to demonstrate theoretical feasibility that are too inefficient to be used in practice, or assumes synchrony, i.e., relies on known bounds on message delays and process speeds. The systems closest to ours, Rampart [30] and SecureRing [16], were designed to be practical, but they rely on the synchrony assumption for correctness, which is dangerous in the presence of malicious attacks. An attacker may compromise the safety of a service by delaying non-faulty nodes or the communication between them until they are tagged as faulty and excluded from the replica group. Such a denial-of-service attack is generally easier than gaining control over a non-faulty node.

Our algorithm is not vulnerable to this type of attack because it does not rely on synchrony for safety. In addition, it improves the performance of Rampart and SecureRing by more than an order of magnitude as explained in Section 7. It uses only one message round trip to execute read-only operations and two to execute read-write operations. Also, it uses an efficient authentication scheme based on message authentication codes during normal operation; public-key cryptography, which was cited as the major latency [29] and throughput [6] problem in Rampart, is used only in view changes.

To evaluate our approach, we implemented a replication library and used it to implement a real service: a Byzantine-fault-tolerant distributed file system that supports the NFS protocol. We used the Andrew benchmark [15] to evaluate the performance of our system. The results show that our system is only 3% slower than the standard NFS daemon in the Digital Unix kernel during normal-case operation.

Thus, the paper makes the following contributions:

- It describes the first state-machine replication protocol that correctly survives Byzantine faults in asynchronous networks.
- It describes a number of important optimizations that allow the algorithm to perform well so that it can be used in real systems.

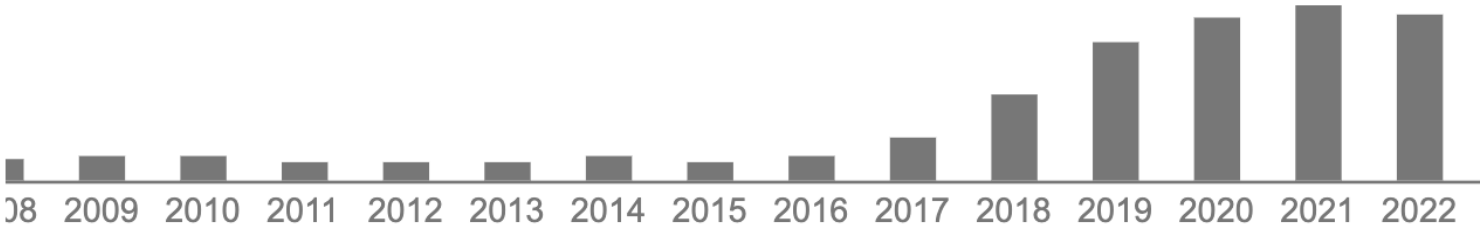
This research was supported by grants No. DARPA under contract DABT63-95-C-005, monitored by Army Fort Huachuca, and under contract F30602-98-1-0237, monitored by the Air Force Research Laboratory, and grant No. NSF. Miguel Castro was partially supported by a PRAXIS XXI fellowship.

- Caveats:
  - set of machines running consensus are well-known (vetted, closed membership)
  - no more than a fraction (1/3) of the machines collude to break the system
- Next lecture, we will start studying BFT consensus, based on a closed membership
- Later in the course we will understand how open membership blockchains work

PBFT citation count revisited

740 citations in 2022

Total citations Cited by 5010



Scholar articles Practical byzantine fault tolerance
M Castro, B Liskov - OsDI, 1999
Cited by 5010 Related articles All 118 versions

Rest of today's lecture

- Revisit the basic cryptographic primitives
- (We will make last year's slides available in fénix, for a more detailed treatment of the subject.)

Definitions

- Information
  - Ordered collection of (binary) symbols, with significance and value
- Entity
  - Party participating in a protocol (person, computer, smart device)
- Attacker (or enemy)
  - Attempts to break security properties of system
- Cryptographic key
  - Collection of numerical parameters, may be known only by one or more entities
  - In particular, we assume the attacker does not know some of the keys

Cryptographic primitives

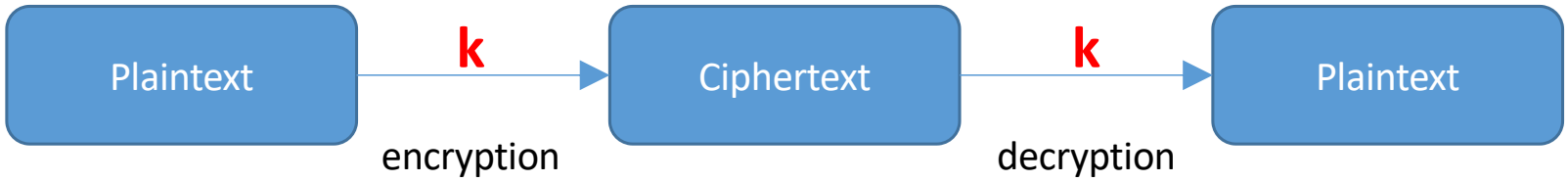
- Symmetric encryption
- Block cipher modes & padding
- Hash functions and MACs
- Asymmetric encryption
- Digital signatures

Kerckhoffs's Principle

- A cryptosystem must be secure despite an attacker that knows all about the system, except the keys
- In other words, security through obscurity doesn't work
- Better to have scrutiny by open experts
  - “The enemy knows the system”
    - Claude Shannon

Symmetric encryption

- Use the same key for encryption and decryption
- Key (**k**) is a shared secret between communicating entities



- Without access to k, cannot obtain Plaintext from Ciphertext

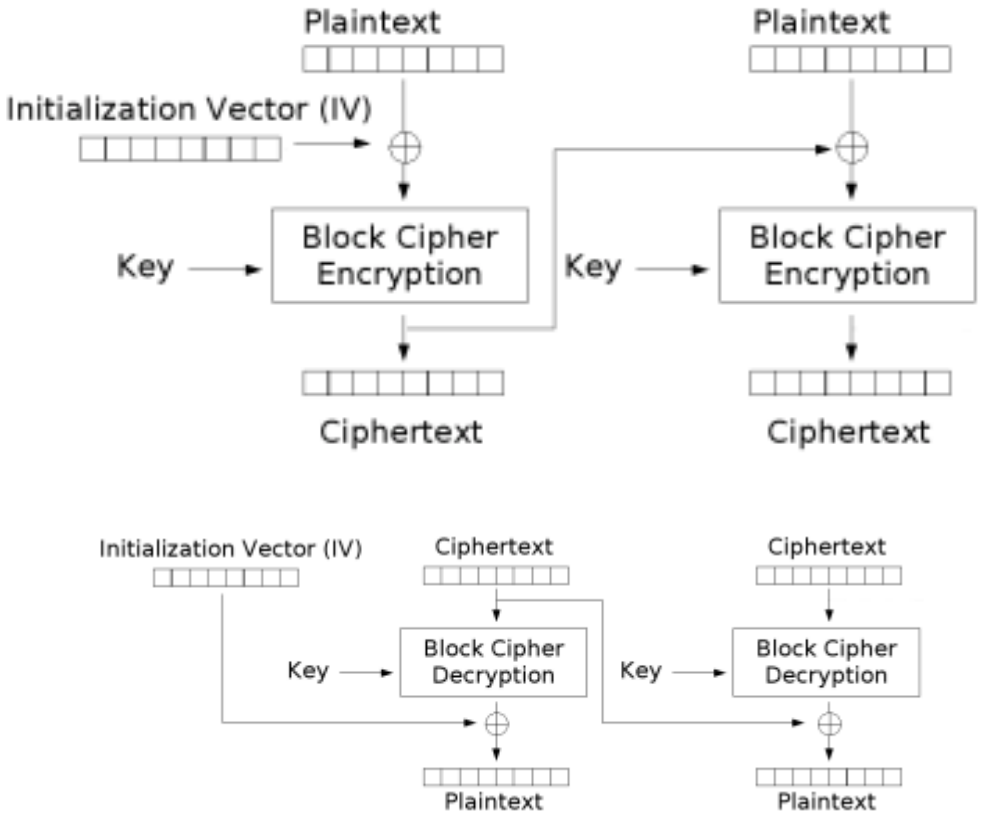
Block ciphers and padding

- Block ciphers encrypt fixed size blocks
  - E.g., AES encrypts 128-bit blocks
- How to encrypt larger blocks or streams of data?
- NIST defines 5 modes of operations:
  - Electronic codebook mode (ECB)
  - Cipher block chaining mode (CBC)
  - Output feedback mode (OFB)
  - Cipher feedback mode (CFB)
  - Counter mode (CTR)

High level idea (explained through CBC mode)

+ Repeated plaintext blocks result in different ciphertext blocks

- A corrupted bit in the ciphertext will affect all subsequent blocks (addressed by other schemes)



Cryptographic Hash Functions

- One-way:
  - For every integer x, easy to compute H(x)
  - Given H(x), hard to find any information about x
- Collision resistance:
  - “Impossible” to find pair (x, y) where $x \neq y$ and H(x) = H(y)
    - 2nd preimage resistance: given x, it is hard to find $y \neq x$ such that H(y) = H(x).
    - (Strong) collision resistance: it is hard to find any x and $y \neq x$ such that H(y) = H(x).

MACs - Message authentication Codes

- Proves authenticity of a message
  - Not modified in transit (integrity) + generated from the correct sender (authenticity)
  - based on a shared secret key between sender and receiver
- Only the entities who hold the key can:
  - Generate the MAC value
  - Check the MAC value
- Does not provide non-repudiation, nor transferrable authentication

Implementing MACs with hashes (HMAC)

- Strawman: Alice and Bob share K, Alice wants to authenticate m and send it to Bob
  - Alice computes HMAC(m,K) = H(K||m)
  - Alice sends <m,HMAC(m,K)>
  - Bob retrieves m and computes H(K||m), compares against received version
  - If they match, message came from Alice (caveat: replay attacks)
- Problem with strawman:
  - Hash functions (e.g., SHA-1) work as state machines, making it easy for an attacker to produce H(K||m||m') from H(K||m)
  - Note: || is the concatenation operator

RFC 2104 - HMAC: Keyed-Hashing for Message Authentication

- HMAC(m,k)= hash(k ⊕ opad || hash(k ⊕ ipad || m))
  - ipad = the byte 0x36 repeated B times
  - opad = the byte 0x5C repeated B times
  - B = block length in bytes

Asymmetric (or public key) encryption

- Each entity has a pair <public key, private key>
- Suppose that Bob wants to send encrypted message to Alice

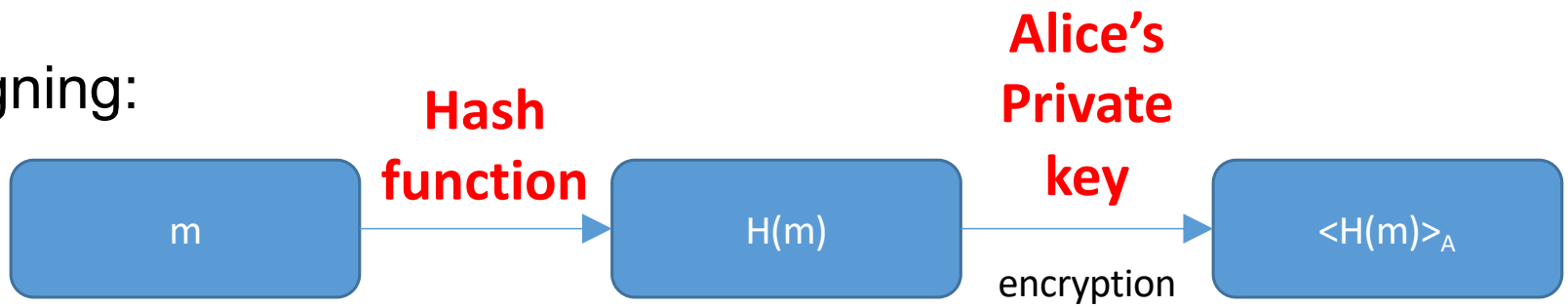
- Without access to Alice's private key, cannot obtain Plaintext from Ciphertext

Digital signatures

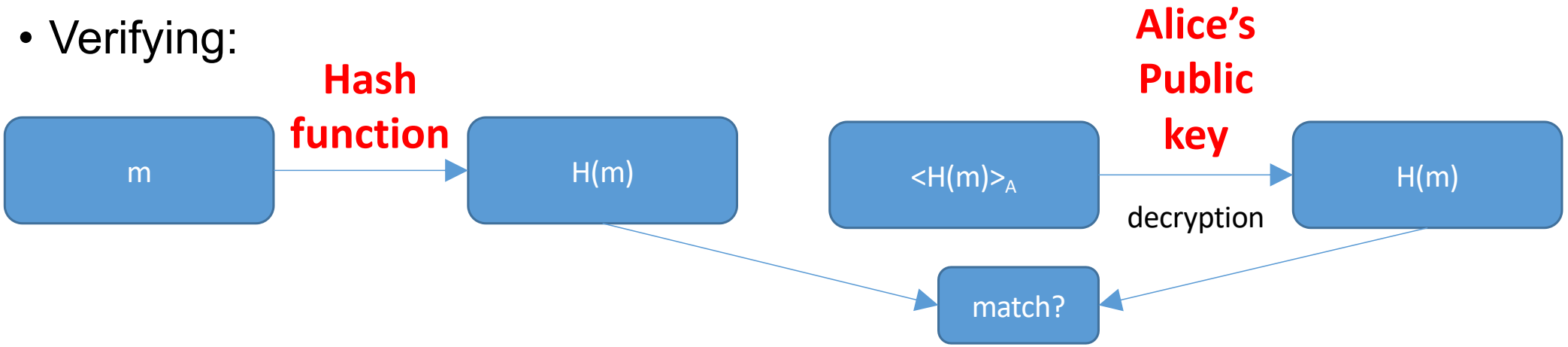
- Provide integrity + authenticity of the author of a message, in a way can be verified by third parties to resolve disputes
  - Non-repudiation property: author cannot claim it did not sign a message (assuming private key was not leaked)

Digital signature scheme

- Signing:



- Alice transmits both m and the signature $<H(m)>_A$
- Verifying:



Next lecture: BFT protocols (building BFT consensus)

- Acknowledgements:
  - Some crypto slides from Ricardo Chaves @ IST, AISS course
  - Some intro slides from Maurice Herlihy @ Brown Univ.