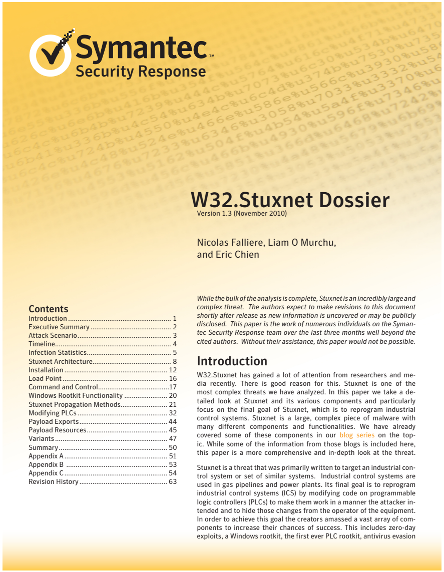
Blockchains: from bitcoin to robotics

Luca Giuzzi

University of Brescia

DMMM Winter School 2020

Why cryptography ?



Symantec™
Security Response

W32.Stuxnet Dossier

Version 1.3 (November 2010)

Nicolas Falliere, Liam O Murchu,
and Eric Chien

Contents

While the bulk of the analysis is complete, Stuxnet is an incredibly large and complex threat. The authors expect to make revisions to this document shortly after release as new information is uncovered or may be publicly disclosed. This paper is the work of numerous individuals on the Symantec Security Response team over the last three months well beyond the cited authors. Without their assistance, this paper would not be possible.

Introduction

W32.Stuxnet has gained a lot of attention from researchers and media recently. There is good reason for this. Stuxnet is one of the most complex threats we have analyzed. In this paper we take a detailed look at Stuxnet and its various components and particularly focus on the final goal of Stuxnet, which is to reprogram industrial control systems. Stuxnet is a large, complex piece of malware with many different components and functionalities. We have already covered some of these components in our blog series on the topic. While some of the information from those blogs is included here, this paper is a more comprehensive and in-depth look at the threat.

Stuxnet is a threat that was primarily written to target an industrial control system or set of similar systems. Industrial control systems are used in gas pipelines and power plants. Its final goal is to reprogram industrial control systems (ICS) by modifying code on programmable logic controllers (PLCs) to make them work in a manner the attacker intended and to hide those changes from the operator of the equipment. In order to achieve this goal the creators amassed a vast array of components to increase their chances of success. This includes zero-day exploits, a Windows rootkit, the first ever PLC rootkit, antivirus evasion

Stuxnet Worm Impact on Industrial Cyber-Physical System Security

Stamatis Karnouskos
SAP Research, Germany
Email: stamatis.karnouskos@sap.com

Abstract—**Industrial systems consider only partially security, mostly relying on the basis of "isolated" networks, and controlled access environments. Monitoring and control systems such as SCADA/DCS are responsible for managing critical infrastructures operate in these environments, where a false sense of security assumptions is usually made. The Stuxnet worm attack demonstrated widely in mid 2010 that many of the security assumptions made about the operating environment, technological capabilities and potential threat risk analysis are far away from the reality and challenges modern industrial systems face. We investigate in this work the highly sophisticated aspects of Stuxnet, the impact that it may have on existing security considerations and pose some thoughts on the next generation SCADA/DCS systems from a security perspective.**

I. Introduction

Much of our critical infrastructure is controlled by cyber-physical systems responsible for monitoring and controlling various processes [1]. The Supervisory Control And Data Acquisition (SCADA) system are industrial control systems responsible for a wide range of industrial processes e.g. manufacturing, power generation, refining, as well as infrastructure e.g. water management, oil & gas pipelines, wind farms, and facilities e.g. airports, space stations, buildings etc. The importance of monitoring and control, which heavily relies on such cyber-physical systems, is paramount for European and world economies in various industrial sectors; indicatively this market is expected to grow from an estimated 275 € Bn in 2012 to 500 € Bn in 2020 [2]. As we move towards large-scale introduction of IT technologies in these sectors, and automatic management, any digital threats that may arise will have a tangible impact on the real world [3] and its processes.

The summer of 2010 was a landmark to the security of the industrial software and equipment industry. By that time it was obvious that a new computer worm called Stuxnet [4] (its name is derived from keywords in its code) was targeting highly specialized industrial systems in critical high-security infrastructures. In the months followed it was becoming clear that this was an unprecedented sophisticated attack that would have wide implications for future industrial systems. For many it was a wakeup call and increased the awareness on security which is still seen as an afterthought and add-on, and not as a continuous process that should be integrated in all operational aspects. Although attacks in IT systems are not something new, up to now it was considered highly unlikely that large scale attacks in the software side of highly specialized applications (such as that of a SCADA) were worth trying or even possible (mainly due to the very niche technology and expertise needed). Additionally it was considered that a "safe" environment (implying disconnected from the Internet and with limited personnel access) was good enough protection. All of these considerations though have been radically changed the last months due to the Stuxnet incident.

This attack comes at an extremely critical time, as modern industrial systems move towards the adoption of Internet based technologies and architectures [5]; although not necessarily connected to the Internet itself. General purpose computing systems, complex industrial applications composable of heterogeneous software and hardware components, wireless access points, abstraction of hardware and uniform access via web services etc. are on the rise. Additionally the enterprise IT systems are getting more interconnected with the industrial ones in order to make sure that events occurring on the shop-floor can be immediately communicated to the respective business processes. The IT industry is well equipped with risk analysis and security tools, however the same does not hold true for industrial systems and the risks may not be adequately assessed.

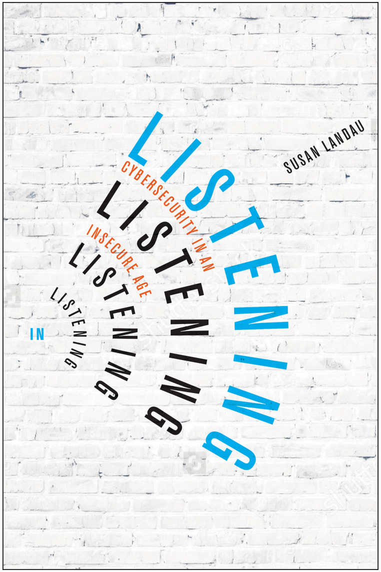
II. The Stuxnet Worm

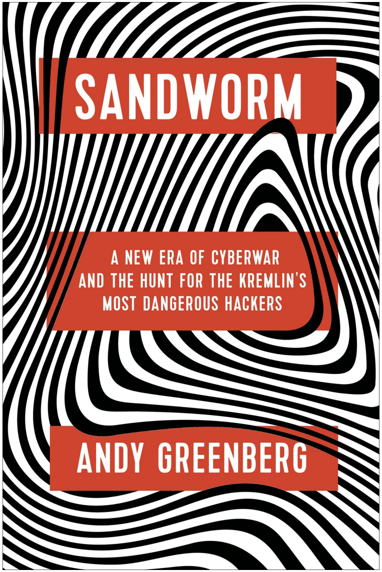
The Stuxnet worm had as its main target industrial control systems with the goal of modifying the code running in Programmable Logic Controllers (PLCs) in order to make them deviate from their expected behavior [6], [7]. This deviation would be small and only noticeable over a longer period of time. In parallel great effort was put by the Stuxnet creators in hiding those changes from the operators, even imitating "legitimate" data. To increase the success rate a vast majority of security holes and tools was used such as rootkits (including what is now known as the first PLC rootkit), antivirus tricking, zero-day exploits, network discovery and P2P updates, process injection etc. Many of these are common on modern PCs however the sophistication of the attach was unprecedentedly well-planned and highly customized for specific industrial systems. Recent analysis [7] points out that more than 80% of the infected systems rely mainly in Iran but also in Indonesia and India. Although the main attacks were detected in mid-2010, early variants of the Stuxnet code stemming from 2009 have been found. It is believed that the development of such a highly sophisticated worm was a joint-effort with experts from different specializations and a huge investment in time and cost.

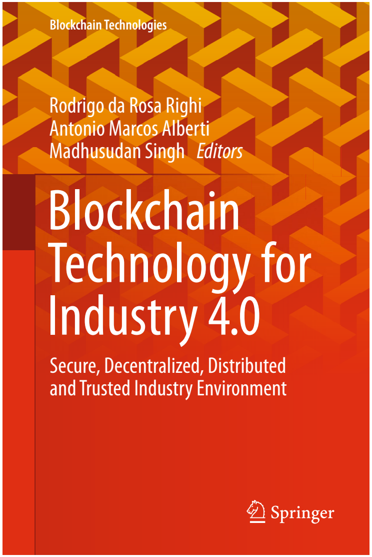
Why ?

- Integrity
- Authentication
- Secure Collaboration
- etc.

Why cryptography ?




SANDWORM
A NEW ERA OF CYBERWAR AND THE HUNT FOR THE KREMLIN'S MOST DANGEROUS HACKERS
ANDY GREENBERG


Blockchain Technologies
Rodrigo da Rosa Righi
Antonio Marcos Alberti
Madhusudan Singh Editors
Blockchain Technology for Industry 4.0
Secure, Decentralized, Distributed and Trusted Industry Environment
Springer

This book series aims to provide details of blockchain implementation in technology and interdisciplinary fields such as Medical Science, Applied Mathematics, Environmental Science, Business Management, and Computer Science. It covers an in-depth knowledge of blockchain technology for advance and emerging future technologies. It focuses on the Magnitude: scope, scale & frequency, Risk: security, reliability trust, and accuracy, Time: latency & timelines, utilization and implementation details of blockchain technologies. While Bitcoin and cryptocurrency might have been the first widely known uses of blockchain technology, but today, it has far many applications. In fact, blockchain is revolutionizing almost every industry. Blockchain has emerged as a disruptive technology, which has not only laid the foundation for all crypto-currencies, but also provides beneficial solutions in other fields of technologies. The features of blockchain technology include decentralized and distributed secure ledgers, recording transactions across a peer-to-peer network, creating the potential to remove unintended errors by providing transparency as well as accountability. This could affect not only the finance technology (crypto-currencies) sector, but also other fields such as:

Crypto-economics Blockchain
Enterprise Blockchain
Blockchain Travel Industry
Embedded Privacy Blockchain
Blockchain Industry 4.0
Blockchain Smart Cities,
Blockchain Future technologies,
Blockchain Fake news Detection,
Blockchain Technology and It's Future Applications
Implications of Blockchain technology
Blockchain Privacy
Blockchain Mining and Use cases
Blockchain Network Applications
Blockchain Smart Contract
Blockchain Architecture
Blockchain Business Models
Blockchain Consensus
Bitcoin and Crypto currencies, and related fields

The initiatives in which the technology is used to distribute and trace the communication start point, provide and manage privacy, and create trustworthy environment, are just a few examples of the utility of blockchain technology, which also highlight the risks, such as privacy protection. Opinion on the utility of blockchain technology has a mixed conception. Some are enthusiastic; others believe that it is merely hyped. Blockchain has also entered the sphere of humanitarian and development aids e.g. supply chain management, digital identity, smart contracts and many more. This book series provides clear concepts and applications of Blockchain technology and invites experts from research centers, academia, industry and government to contribute to it.

If you are interested in contributing to this series, please contact msingh@endicott.ac.kr OR loyola.dsilva@springer.com

More information about this series at http://www.springer.com/series/16276

Blockchains

1. Immutable distributed database
2. Byzantine agreement protocol

Remark

A blockchain might offer other facilities:

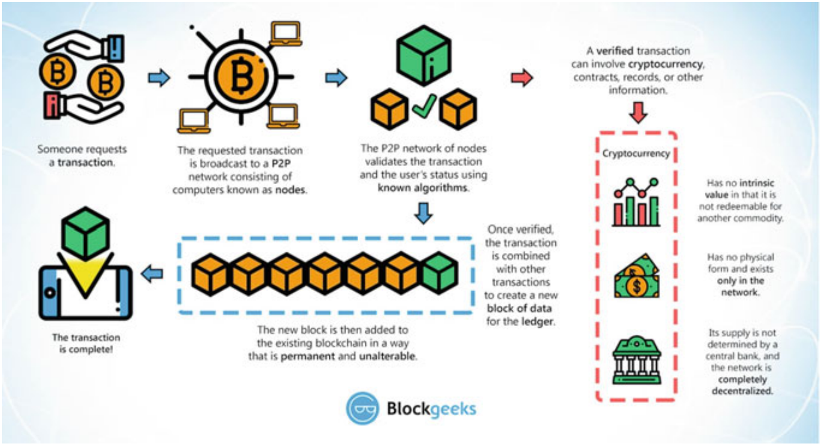
- Virtual Machines
- Smart contracts
- Storage optimizations (Merkle trees)
- etc.

Immutable distributed database

A blockchain is

- Distributed: data can be read and written by a set of non-coordinated agents;
- Immutable: once accepted the data cannot be altered in any way.

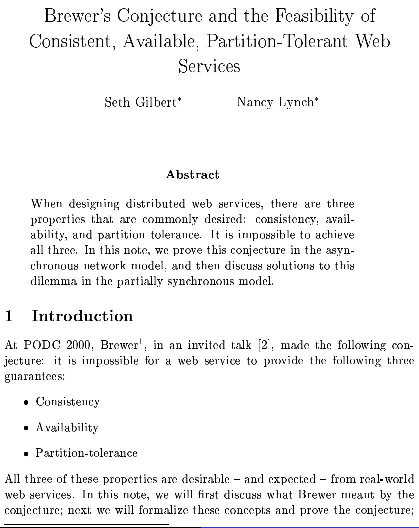
Blockchain


Someone requests a transaction.
The requested transaction is broadcast to a P2P network consisting of computers known as nodes.
The P2P network of nodes validates the transaction and the user's status using known algorithms.
A verified transaction can involve cryptocurrency, contracts, records, or other information.
Cryptocurrency
Has no intrinsic value in that it is not redeemable for another commodity.
Has no physical form and exists only in the network.
Its supply is not determined by a central bank, and the network is completely decentralized.
Once verified, the transaction is combined with other transactions to create a new block of data for the ledger.
The new block is then added to the existing blockchain in a way that is permanent and unalterable.
The transaction is complete!
Blockgeeks

Consensus

- Different agents can have a different view of the database (fork).
- We need an algorithm for deciding which is the version of the database to be trusted in the case of conflicts (byzantine agreement).

Limitations



Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

Seth Gilbert[*] Nancy Lynch[*]

Abstract

When designing distributed web services, there are three properties that are commonly desired: consistency, availability, and partition tolerance. It is impossible to achieve all three. In this note, we prove this conjecture in the asynchronous network model, and then discuss solutions to this dilemma in the partially synchronous model.

1 Introduction

At PODC 2000, Brewer[1], in an invited talk [2], made the following conjecture: it is impossible for a web service to provide the following three guarantees:

- Consistency
- Availability
- Partition-tolerance

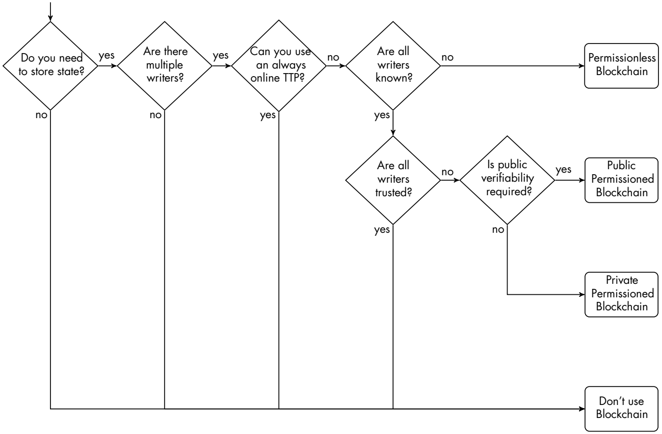
All three of these properties are desirable – and expected – from real-world web services. In this note, we will first discuss what Brewer meant by the conjecture; next we will formalize these concepts and prove the conjecture;

Taxonomy of blockchains

Various kinds:

- Public/Private
- Permissioned/Permissionless
- Decentralized/Centralized

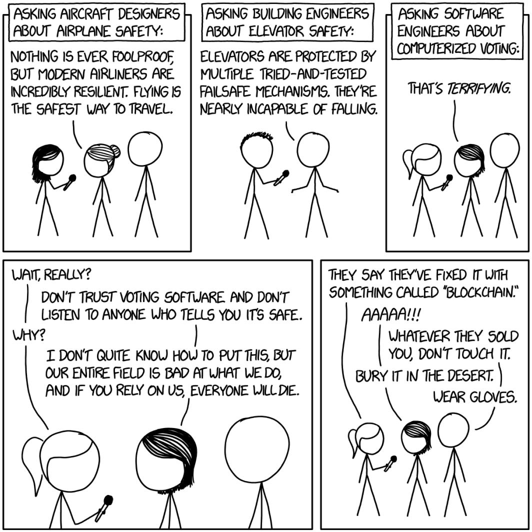
Taxonomy of blockchains/2


Do you need to store state?
yes
no
Are there multiple writers?
yes
no
Can you use an always online TTP?
no
yes
Are all writers known?
no
yes
Permissionless Blockchain
Are all writers trusted?
no
yes
Is public verifiability required?
yes
no
Public Permissioned Blockchain
Private Permissioned Blockchain
Don't use Blockchain

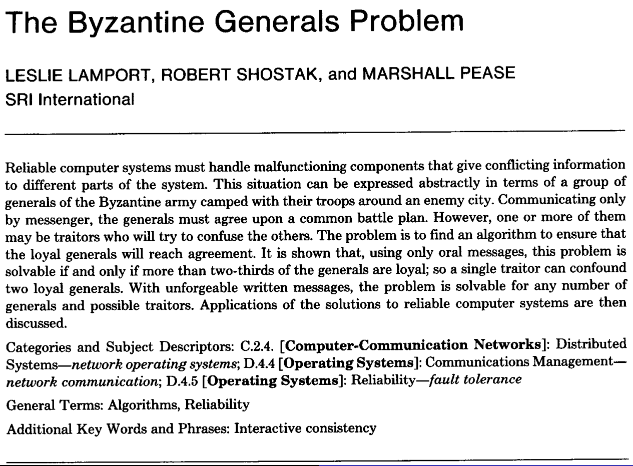
Applications

- Electronic currencies (bitcoin, ethereum, libra, etc.)
- Smart contracts (ethereum, libra, hyperledger)
- Asset tracking (hyperledger)
- Manufacturing quality control
- E-voting (?)
- etc.

Uses of blockchains


ASKING AIRCRAFT DESIGNERS ABOUT AIRPLANE SAFETY:
NOTHING IS EVER FOOLPROOF, BUT MODERN AIRLINERS ARE INCREDIBLY RESILIENT. FLYING IS THE SAFEST WAY TO TRAVEL.
ASKING BUILDING ENGINEERS ABOUT ELEVATOR SAFETY:
ELEVATORS ARE PROTECTED BY MULTIPLE TRIED-AND-TESTED FAILSAFE MECHANISMS. THEY'RE NEARLY INCAPABLE OF FALLING.
ASKING SOFTWARE ENGINEERS ABOUT COMPUTERIZED VOTING:
THAT'S TERRIFYING.
WAIT, REALLY?
DON'T TRUST VOTING SOFTWARE AND DON'T LISTEN TO ANYONE WHO TELLS YOU IT'S SAFE.
WHY?
I DON'T QUITE KNOW HOW TO PUT THIS, BUT OUR ENTIRE FIELD IS BAD AT WHAT WE DO, AND IF YOU RELY ON US, EVERYONE WILL DIE.
THEY SAY THEY'VE FIXED IT WITH SOMETHING CALLED "BLOCKCHAIN."
AAAAA!!!
WHATEVER THEY SOLD YOU, DON'T TOUCH IT.
BURY IT IN THE DESERT.
WEAR GLOVES.

Byzantine agreement



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 Systems—*network operating systems*; D.4.4 **[Operating Systems]**: Communications Management—*network communication*; D.4.5 **[Operating Systems]**: Reliability—*fault tolerance*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Interactive consistency

Byzantine agreement

A byzantine agreement protocol is a distributed protocol among agents $A_1, \ldots, A_k$ such that

1. all non-faulty/honest agents terminate the protocol in a finite number of steps;
2. all non-faulty/honest agents agree upon termination on the same decision;
3. different decisions are possible.

Note

The decision taken is *irrelevant* as far as all non-faulty agents agree.

Byzantine fault

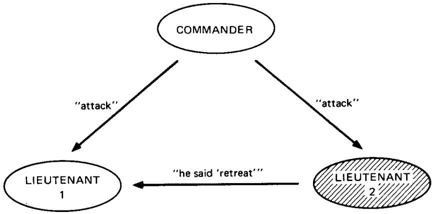


Fig. 1. Lieutenant 2 a traitor.

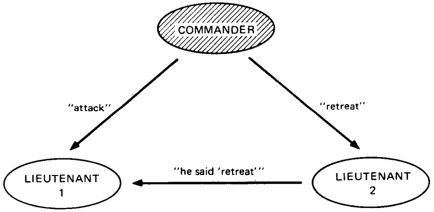


Fig. 2. The commander a traitor.

Byzantine agreement

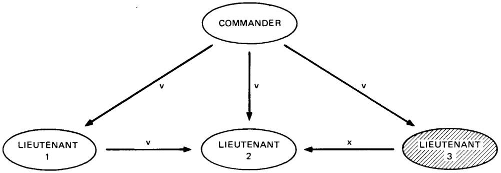


Fig. 3. Algorithm OM(1); Lieutenant 3 a traitor.

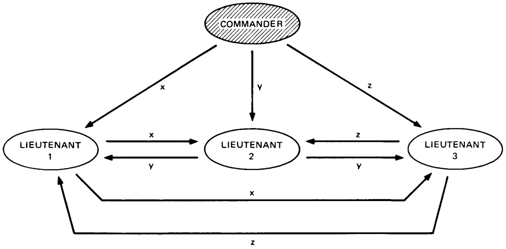
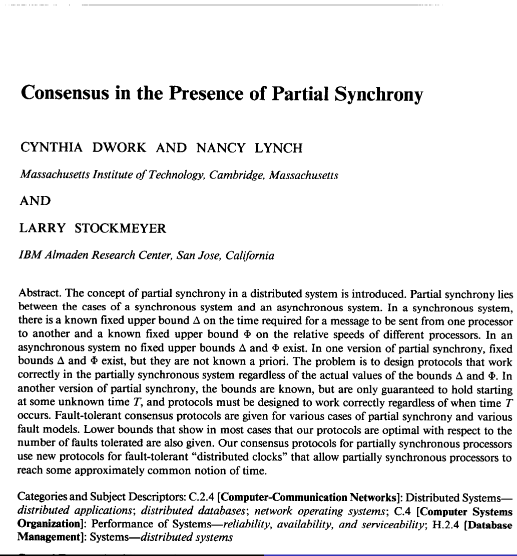


Fig. 4. Algorithm OM(1); the commander a traitor.

Byzantine agreement/limits



Consensus in the Presence of Partial Synchrony

CYNTHIA DWORK AND NANCY LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

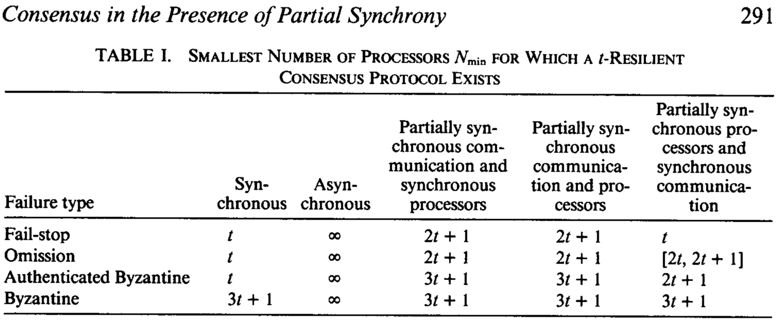
LARRY STOCKMEYER

IBM Almaden Research Center, San Jose, California

Abstract. The concept of partial synchrony in a distributed system is introduced. Partial synchrony lies between the cases of a synchronous system and an asynchronous system. In a synchronous system, there is a known fixed upper bound Δ on the time required for a message to be sent from one processor to another and a known fixed upper bound Φ on the relative speeds of different processors. In an asynchronous system no fixed upper bounds Δ and Φ exist. In one version of partial synchrony, fixed bounds Δ and Φ exist, but they are not known a priori. The problem is to design protocols that work correctly in the partially synchronous system regardless of the actual values of the bounds Δ and Φ. In another version of partial synchrony, the bounds are known, but are only guaranteed to hold starting at some unknown time T, and protocols must be designed to work correctly regardless of when time T occurs. Fault-tolerant consensus protocols are given for various cases of partial synchrony and various fault models. Lower bounds that show in most cases that our protocols are optimal with respect to the number of faults tolerated are also given. Our consensus protocols for partially synchronous processors use new protocols for fault-tolerant "distributed clocks" that allow partially synchronous processors to reach some approximately common notion of time.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; *distributed databases*; *network operating systems*; C.4 [**Computer Systems Organization**]: Performance of Systems—*reliability, availability, and serviceability*; H.2.4 [**Database Management**]: Systems—*distributed systems*

Byzantine agreement/limits

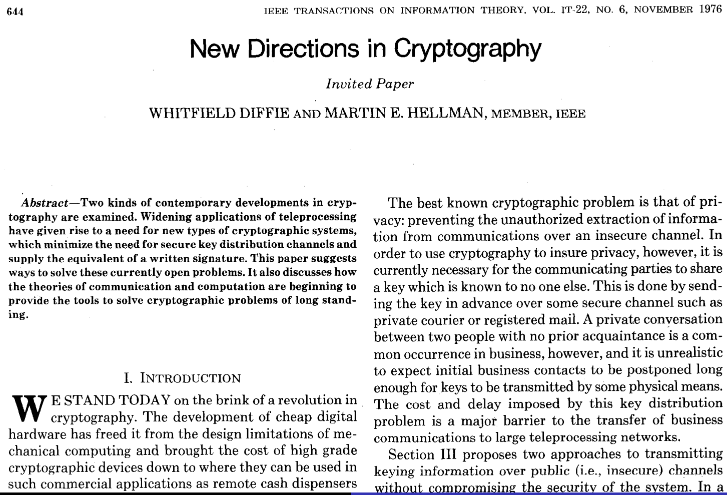


Consensus in the Presence of Partial Synchrony 291

TABLE I. SMALLEST NUMBER OF PROCESSORS $N_{\min}$ FOR WHICH A t-RESILIENT CONSENSUS PROTOCOL EXISTS

Failure type	Synchronous	Asynchronous	Partially synchronous communication and synchronous processors	Partially synchronous communication and processors	Partially synchronous processors and synchronous communication
Fail-stop	t	∞	$2t + 1$	$2t + 1$	t
Omission	t	∞	$2t + 1$	$2t + 1$	$[2t, 2t + 1]$
Authenticated Byzantine	t	∞	$3t + 1$	$3t + 1$	$2t + 1$
Byzantine	$3t + 1$	∞	$3t + 1$	$3t + 1$	$3t + 1$

Digital signatures



644 IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-22, NO. 6, NOVEMBER 1976

New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

***Abstract*—Two kinds of contemporary developments in cryptography are examined. Widening applications of teleprocessing have given rise to a need for new types of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. It also discusses how the theories of communication and computation are beginning to provide the tools to solve cryptographic problems of long standing.**

I. INTRODUCTION

WE STAND TODAY on the brink of a revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high grade cryptographic devices down to where they can be used in such commercial applications as remote cash dispensers

The best known cryptographic problem is that of privacy: preventing the unauthorized extraction of information from communications over an insecure channel. In order to use cryptography to insure privacy, however, it is currently necessary for the communicating parties to share a key which is known to no one else. This is done by sending the key in advance over some secure channel such as private courier or registered mail. A private conversation between two people with no prior acquaintance is a common occurrence in business, however, and it is unrealistic to expect initial business contacts to be postponed long enough for keys to be transmitted by some physical means. The cost and delay imposed by this key distribution problem is a major barrier to the transfer of business communications to large teleprocessing networks.

Section III proposes two approaches to transmitting keying information over public (i.e., insecure) channels without compromising the security of the system. In a

Digital signatures

- Informally: electronic equivalent of a signature; can be used to authorize or authenticate operations.
- Formally: three algorithms $\mathtt{Gen}$, $\mathtt{Sign}$, $\mathtt{Verify}$ such that
    1. $\mathtt{Gen}$: takes as input a security parameter k and returns a pair of keys $(sk, pk) \leftarrow \mathtt{Gen}(k)$.
    2. $\mathtt{Sign}$: takes as input a security parameter k, the secret key sk and a message $m \in M_k$ and outputs a signature $\sigma \leftarrow \mathtt{Sign}_{sk}(m)$
    3. $\mathtt{Verify}$: takes as input a public key pk, a message m and a purpoted signature σ and returns a bit $b = \mathtt{Verify}_{pk}(m, \sigma)$.
    4. $\forall (sk, pk) \leftarrow \mathtt{Gen}(k), \forall m \in M_k, \forall \sigma \leftarrow \mathtt{Sign}_{sk}(m)$,
    $$\mathtt{Verify}_{pk}(m, \sigma) = 1.$$

Digital signatures/security

A signature is

- Existentially unforgeable if for all probabilistic polynomial time adversaries A the following is negligible

$$\varepsilon_A(k) = \Pr\left[\begin{array}{lcc} \{m_i\}_{i=1}^{\ell} \leftarrow M_k; (pk, sk) \leftarrow \mathtt{Gen}(k) & & \mathtt{Verify}_{pk}(m,\sigma) = 1 \\ \forall i \in \{1,\dots,\ell\} : \sigma_i \leftarrow \mathtt{Sign}_{sk}(m_i) & : & \text{and} \\ (m,\sigma) \leftarrow A(pk, \{m_i,\sigma_i\}_{i=1}^{\ell}) & & m \notin \{m_i\}_{i=1}^{\ell} \end{array}\right]$$

- Strongly unforgeable if for all probabilistic polynomial time adversaries A the following is negligible

$$\varepsilon_A(k) = \Pr\left[\begin{array}{lcc} \{m_i\}_{i=1}^{\ell} \leftarrow M_k; (pk, sk) \leftarrow \mathtt{Gen}(k) & & \mathtt{Verify}_{pk}(m,\sigma) = 1 \\ \forall i \in \{1,\dots,\ell\} : \sigma_i \leftarrow \mathtt{Sign}_{sk}(m_i) & : & \text{and} \\ (m,\sigma) \leftarrow A(pk, \{m_i,\sigma_i\}_{i=1}^{\ell}) & & (m,\sigma) \notin \{(m_i,\sigma_i)\}_{i=1}^{\ell} \end{array}\right]$$

Digital signatures/security

- *sk* is the secret key and it is private
- *pk* is the public key and it is public
- only the owner of *sk* can sign a message *m*; so a correct signature on *m* attests that the owner of *sk* has seen/read/endorsed *m*
- everybody can verify the signature on a message

We get:

1. Authentication
2. Non-repudiation

Byzantine agreement with signatures

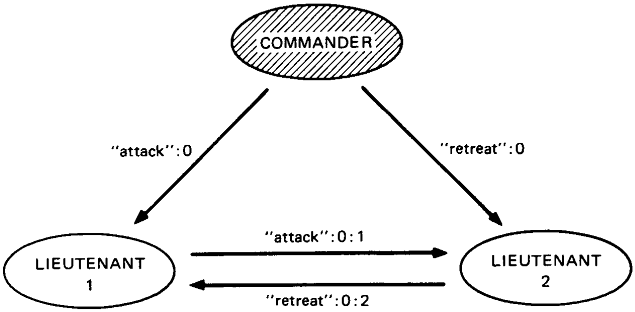
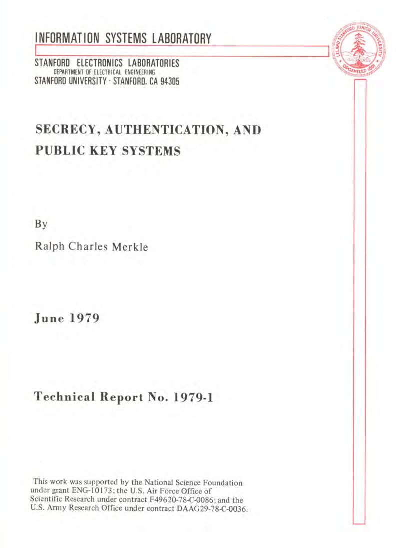


Fig. 5. Algorithm SM(1); the commander a traitor.

Hash functions



INFORMATION SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES
DEPARTMENT OF ELECTRICAL ENGINEERING
STANFORD UNIVERSITY · STANFORD, CA 94305

LELAND STANFORD JUNIOR UNIVERSITY · ORGANIZED 1891

SECRECY, AUTHENTICATION, AND PUBLIC KEY SYSTEMS

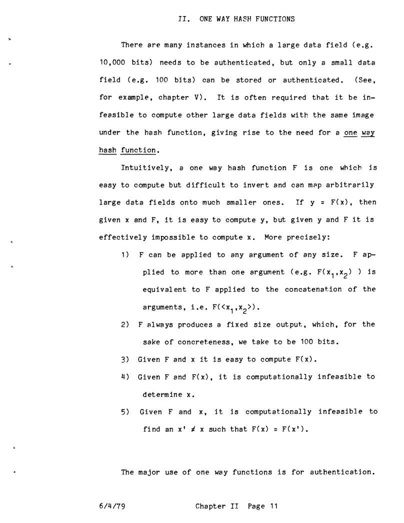
By

Ralph Charles Merkle

June 1979

Technical Report No. 1979-1

This work was supported by the National Science Foundation under grant ENG-10173; the U.S. Air Force Office of Scientific Research under contract F49620-78-C-0086; and the U.S. Army Research Office under contract DAAG29-78-C-0036.



II. ONE WAY HASH FUNCTIONS

There are many instances in which a large data field (e.g. 10,000 bits) needs to be authenticated, but only a small data field (e.g. 100 bits) can be stored or authenticated. (See, for example, chapter V). It is often required that it be infeasible to compute other large data fields with the same image under the hash function, giving rise to the need for a one way hash function.

Intuitively, a one way hash function F is one which is easy to compute but difficult to invert and can map arbitrarily large data fields onto much smaller ones. If y = F(x), then given x and F, it is easy to compute y, but given y and F it is effectively impossible to compute x. More precisely:

1) F can be applied to any argument of any size. F applied to more than one argument (e.g. $F(x_1,x_2)$) is equivalent to F applied to the concatenation of the arguments, i.e. $F(<x_1,x_2>)$.

2) F always produces a fixed size output, which, for the sake of concreteness, we take to be 100 bits.

3) Given F and x it is easy to compute F(x).

4) Given F and F(x), it is computationally infeasible to determine x.

5) Given F and x, it is computationally infeasible to find an $x' \neq x$ such that $F(x) = F(x')$.

The major use of one way functions is for authentication.

6/4/79 Chapter II Page 11

Hash functions (MDC)

$$h : \{0,1\}^* \to \{0,1\}^k$$

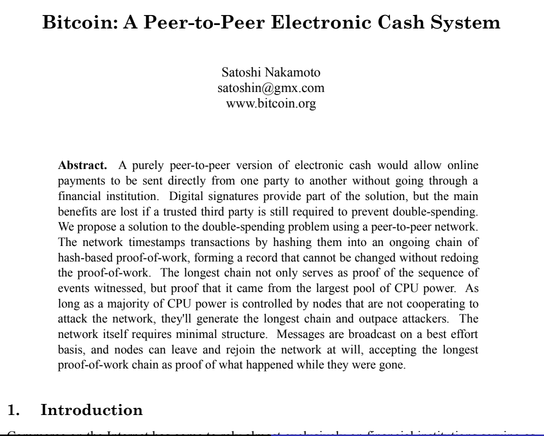
such that it is computationally infeasible to:

1. given $\mathbf{c} \in \{0,1\}^k$ determine $\mathbf{x} \in \{0,1\}^*$ such that $h(\mathbf{x}) = \mathbf{c}$;
2. given $\mathbf{x} \in \{0,1\}^*$ determine $\mathbf{y} \in \{0,1\}^*$ with $\mathbf{x} \neq \mathbf{y}$ and $h(\mathbf{x}) = h(\mathbf{y})$;
3. determine $\mathbf{x}, \mathbf{y} \in \{0,1\}^*$ such that $h(\mathbf{x}) = h(\mathbf{y})$.

Hash functions (MDC)

- Provide *digests* of messages to simplify signatures
- Provide a way to construct *robust pointers*
- Provide problems which are computationally expensive to solve

Bitcoin



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.

1. Introduction

Bitcoin

- Electronic cash not baked by external entities
- Based on a blockchain which is
  1. Public
  2. Permissionless
  3. Distributed

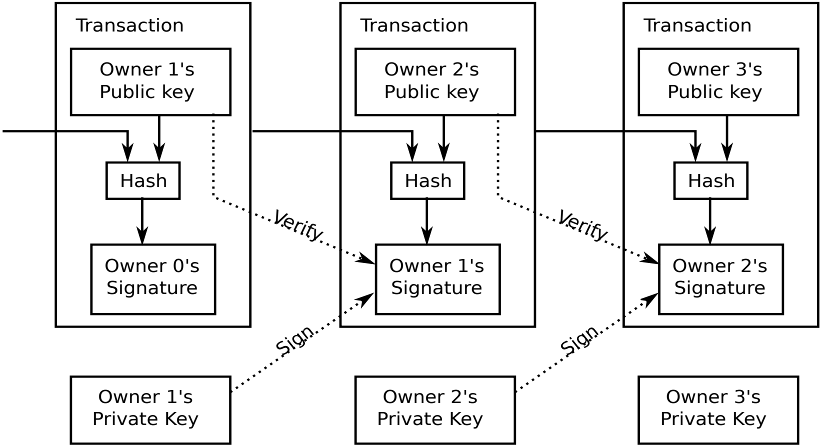
Bitcoin/2

- Fully decentralized and peer-to-peer;
- Database: list of all transactions between pseudonimous accounts;
- Blocks are linked by means of hash functions acting as pointers;
- Transactions are validated by means of digital signatures;
- Consensus is reached by proof-of-work.

Bitcoin/transactions

- People in bitcoin are identified by their public keys.
- Suppose Alice to have keys (pk_A, sk_A) and Bob to have keys (pk_B, sk_B).
- A payment form Alice to Bob is a message *m* containing as payee pk_B signed with sk_A and indicating a certain number of bitcoins to be transferred.
- The database is updated by subtracting the number of bitcoins paid by Alice from the account pk_A and crediting the same amount on the account pk_B.
- Digital signatures guarantee authentication.
- Transactions are batched in *blocks*.

Transactions



Bitcoin/consensus: proof of work

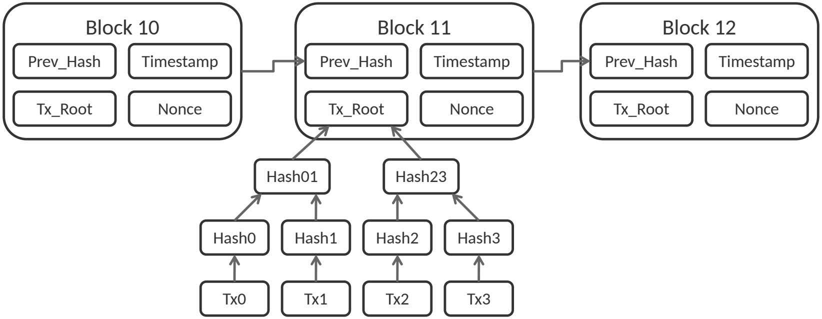
How to guarantee consensus? (against malicious agents or errors)

- All transaction blocks must be appended to a linear chain (blockchain).
- Appending blocks is expensive.
- If there is a fork (more than one potentially valid chain) all honest agents must choose the longest chain.
- Ultimately all honest agents will reach an agreement on transactions *deep enough*.
- We want *consistency* and do not care about truth.

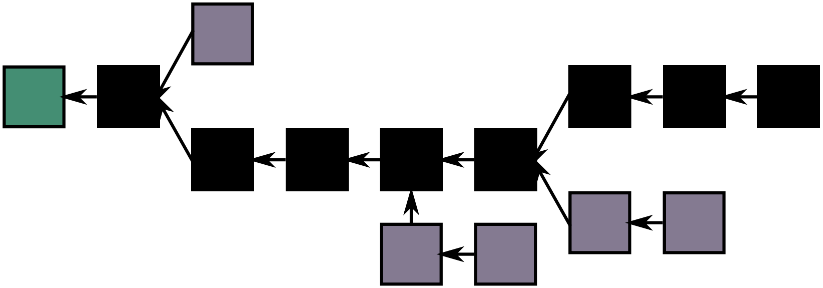
Note

We are not interested in whether Alice has *really* paid Bob or not but we want that for all agents it is true that either Alice has been debited and Bob credited or Alice has not been debited and Bob has not been credited.

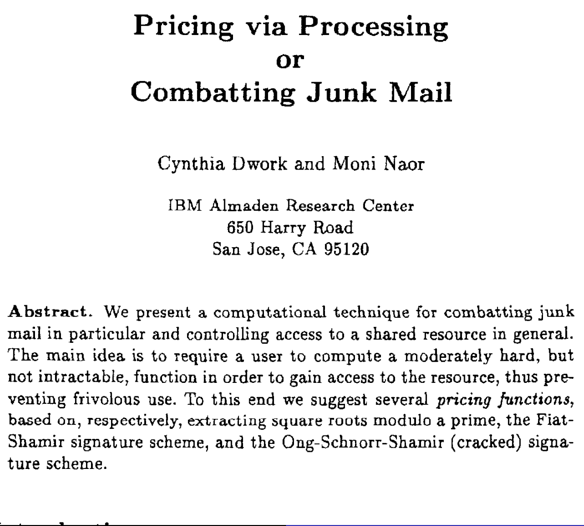
Mining


Block 10
Prev_Hash
Timestamp
Tx_Root
Nonce
Block 11
Prev_Hash
Timestamp
Tx_Root
Nonce
Block 12
Prev_Hash
Timestamp
Tx_Root
Nonce
Hash01
Hash23
Hash0
Hash1
Hash2
Hash3
Tx0
Tx1
Tx2
Tx3

Forks



Proof of Work



Pricing via Processing or Combatting Junk Mail

Cynthia Dwork and Moni Naor

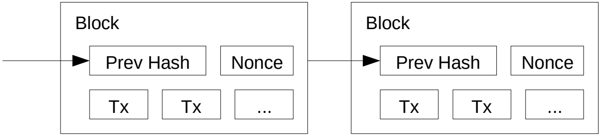
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

Abstract. We present a computational technique for combatting junk mail in particular and controlling access to a shared resource in general. The main idea is to require a user to compute a moderately hard, but not intractable, function in order to gain access to the resource, thus preventing frivolous use. To this end we suggest several *pricing functions*, based on, respectively, extracting square roots modulo a prime, the Fiat-Shamir signature scheme, and the Ong-Schnorr-Shamir (cracked) signature scheme.

Proof of Work (PoW)

- In order to append a block to the chain it is necessary to solve a computationally hard problem.
- The first to solve the problem has the right to append the block.
- Each participant chooses as valid chain the longest available.
- To alter the contents of a block it would be necessary to solve several PoW problems faster than the growth of the chain.

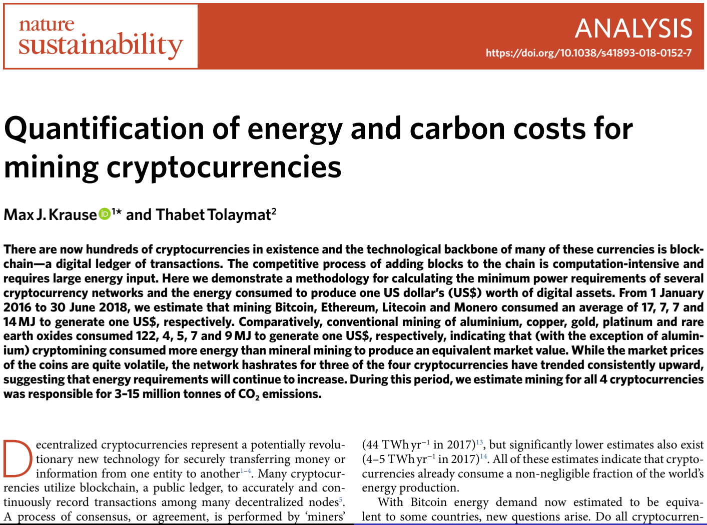
Bitcoin/PoW



Work to be done

- Determine a `Nonce` such that the hash of the *combined block* is less than $N = 2^h$ (for suitable h).

Proof of Work/costs



nature sustainability

ANALYSIS
https://doi.org/10.1038/s41893-018-0152-7

Quantification of energy and carbon costs for mining cryptocurrencies

Max J. Krause[1*] and Thabet Tolaymat[2]

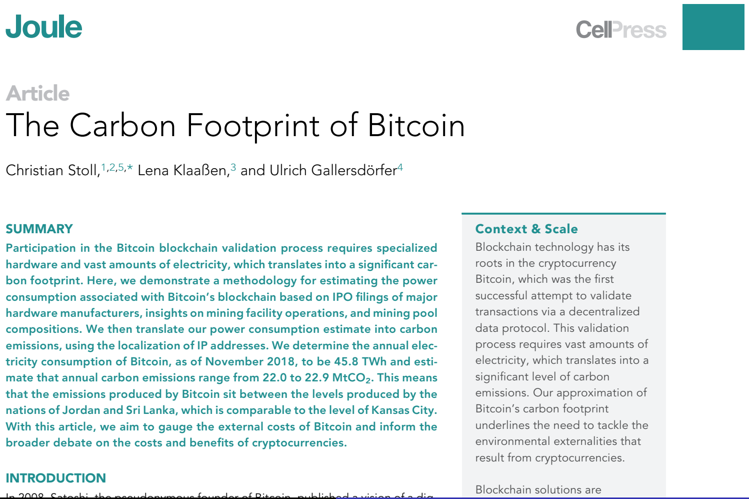
There are now hundreds of cryptocurrencies in existence and the technological backbone of many of these currencies is blockchain—a digital ledger of transactions. The competitive process of adding blocks to the chain is computation-intensive and requires large energy input. Here we demonstrate a methodology for calculating the minimum power requirements of several cryptocurrency networks and the energy consumed to produce one US dollar's (US\$) worth of digital assets. From 1 January 2016 to 30 June 2018, we estimate that mining Bitcoin, Ethereum, Litecoin and Monero consumed an average of 17, 7, 7 and 14 MJ to generate one US\$, respectively. Comparatively, conventional mining of aluminium, copper, gold, platinum and rare earth oxides consumed 122, 4, 5, 7 and 9 MJ to generate one US\$, respectively, indicating that (with the exception of aluminium) cryptomining consumed more energy than mineral mining to produce an equivalent market value. While the market prices of the coins are quite volatile, the network hashrates for three of the four cryptocurrencies have trended consistently upward, suggesting that energy requirements will continue to increase. During this period, we estimate mining for all 4 cryptocurrencies was responsible for 3–15 million tonnes of CO_2 emissions.

Decentralized cryptocurrencies represent a potentially revolutionary new technology for securely transferring money or information from one entity to another[1–4]. Many cryptocurrencies utilize blockchain, a public ledger, to accurately and continuously record transactions among many decentralized nodes[5]. A process of consensus, or agreement, is performed by 'miners'

(44 TWh yr^{-1} in 2017)[13], but significantly lower estimates also exist (4–5 TWh yr^{-1} in 2017)[14]. All of these estimates indicate that cryptocurrencies already consume a non-negligible fraction of the world's energy production.

With Bitcoin energy demand now estimated to be equivalent to some countries, new questions arise. Do all cryptocurren-

Proof of Work/costs



Joule CellPress

Article

The Carbon Footprint of Bitcoin

Christian Stoll,[1,2,5,*] Lena Klaaßen,[3] and Ulrich Gallersdörfer[4]

SUMMARY

Participation in the Bitcoin blockchain validation process requires specialized hardware and vast amounts of electricity, which translates into a significant carbon footprint. Here, we demonstrate a methodology for estimating the power consumption associated with Bitcoin's blockchain based on IPO filings of major hardware manufacturers, insights on mining facility operations, and mining pool compositions. We then translate our power consumption estimate into carbon emissions, using the localization of IP addresses. We determine the annual electricity consumption of Bitcoin, as of November 2018, to be 45.8 TWh and estimate that annual carbon emissions range from 22.0 to 22.9 $MtCO_2$. This means that the emissions produced by Bitcoin sit between the levels produced by the nations of Jordan and Sri Lanka, which is comparable to the level of Kansas City. With this article, we aim to gauge the external costs of Bitcoin and inform the broader debate on the costs and benefits of cryptocurrencies.

Context & Scale

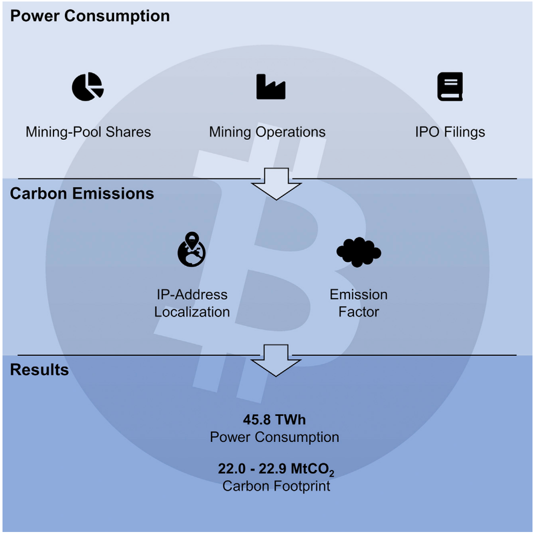
Blockchain technology has its roots in the cryptocurrency Bitcoin, which was the first successful attempt to validate transactions via a decentralized data protocol. This validation process requires vast amounts of electricity, which translates into a significant level of carbon emissions. Our approximation of Bitcoin's carbon footprint underlines the need to tackle the environmental externalities that result from cryptocurrencies.

Blockchain solutions are

INTRODUCTION

In 2008, Satoshi, the pseudonymous founder of Bitcoin, published a vision of a dig-

Proof of Work/costs



Proof of Work/considerations

Remark

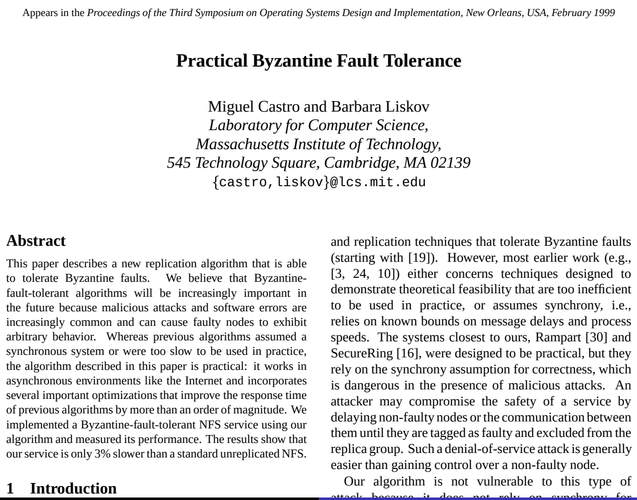
A Byzantine attacker in bitcoin aims to keep a fork alive; ultimately this costs more than the expected gain.

- Mining works well for e-currencies
- It does not work so well for:
  1. Tracking external items
  2. Validating code to be executed
  3. Enforcing fairness

P2P consensus algorithms

- Proof of Work
- Proof of Burn
- Proof of Stake
- Proof of Authority (endorsement)

Practical Byzantine Fault tolerance



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 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

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., 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

Practical Byzantine Fault tolerance

Several phases:

1. Request: a client sends a request for an update to a primary;
2. Pre-Prepare: the primary notifies the backups that has received a request for a given view *v* and assigns a request number;
3. Prepare: all backups which accepted pre-prepare enter the prepare phase by multicasting a message with the sequence number and the view number;
4. Commit: once a replica has received a sufficient number of prepare messages sends a commit message to the others;
5. Reply: the primary and the backup reply to the request.

Practical Byzantine Fault tolerance

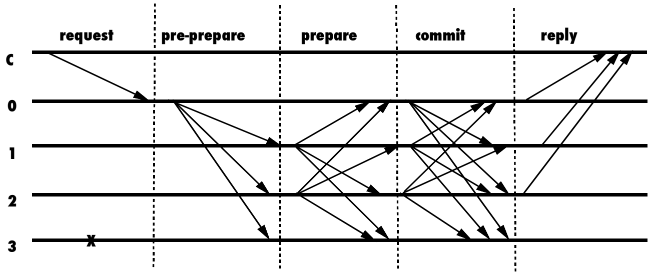
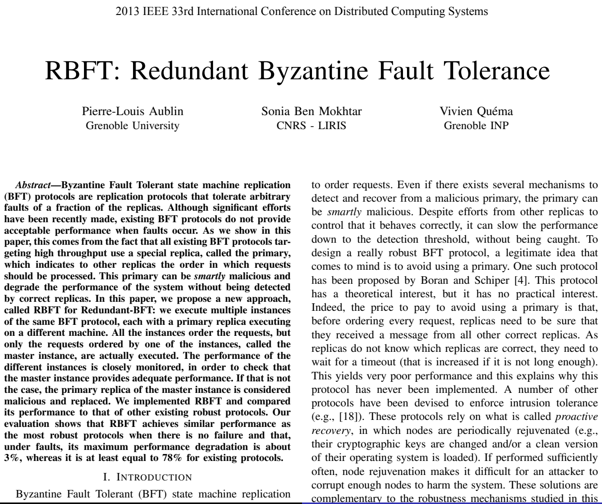


Figure 1: Normal Case Operation

Redundant Byzantine Fault tolerance



2013 IEEE 33rd International Conference on Distributed Computing Systems

RBFT: Redundant Byzantine Fault Tolerance

Pierre-Louis Aublin
Grenoble University

Sonia Ben Mokhtar
CNRS - LIRIS

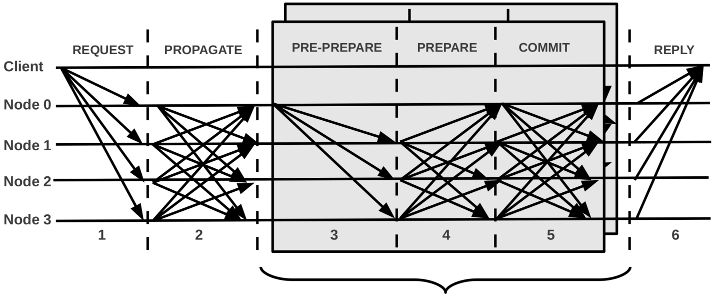
Vivien Quéma
Grenoble INP

***Abstract*—Byzantine Fault Tolerant state machine replication (BFT) protocols are replication protocols that tolerate arbitrary faults of a fraction of the replicas. Although significant efforts have been recently made, existing BFT protocols do not provide acceptable performance when faults occur. As we show in this paper, this comes from the fact that all existing BFT protocols targeting high throughput use a special replica, called the primary, which indicates to other replicas the order in which requests should be processed. This primary can be *smartly* malicious and degrade the performance of the system without being detected by correct replicas. In this paper, we propose a new approach, called RBFT for Redundant-BFT: we execute multiple instances of the same BFT protocol, each with a primary replica executing on a different machine. All the instances order the requests, but only the requests ordered by one of the instances, called the master instance, are actually executed. The performance of the different instances is closely monitored, in order to check that the master instance provides adequate performance. If that is not the case, the primary replica of the master instance is considered malicious and replaced. We implemented RBFT and compared its performance to that of other existing robust protocols. Our evaluation shows that RBFT achieves similar performance as the most robust protocols when there is no failure and that, under faults, its maximum performance degradation is about 3%, whereas it is at least equal to 78% for existing protocols.**

I. Introduction

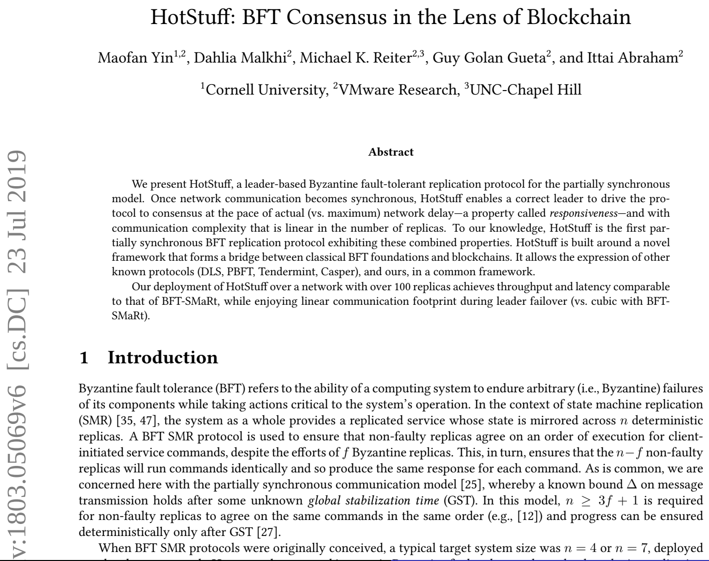
Byzantine Fault Tolerant (BFT) state machine replication to order requests. Even if there exists several mechanisms to detect and recover from a malicious primary, the primary can be *smartly* malicious. Despite efforts from other replicas to control that it behaves correctly, it can slow the performance down to the detection threshold, without being caught. To design a really robust BFT protocol, a legitimate idea that comes to mind is to avoid using a primary. One such protocol has been proposed by Boran and Schiper [4]. This protocol has a theoretical interest, but it has no practical interest. Indeed, the price to pay to avoid using a primary is that, before ordering every request, replicas need to be sure that they received a message from all other correct replicas. As replicas do not know which replicas are correct, they need to wait for a timeout (that is increased if it is not long enough). This yields very poor performance and this explains why this protocol has never been implemented. A number of other protocols have been devised to enforce intrusion tolerance (e.g., [18]). These protocols rely on what is called *proactive recovery*, in which nodes are periodically rejuvenated (e.g., their cryptographic keys are changed and/or a clean version of their operating system is loaded). If performed sufficiently often, node rejuvenation makes it difficult for an attacker to corrupt enough nodes to harm the system. These solutions are complementary to the robustness mechanisms studied in this

Redundant Byzantine Fault tolerance



Redundant agreement performed by the replicas

HotStuff



HotStuff: BFT Consensus in the Lens of Blockchain

Maofan Yin[1,2], Dahlia Malkhi[2], Michael K. Reiter[2,3], Guy Golan Gueta[2], and Ittai Abraham[2]

[1]Cornell University, [2]VMware Research, [3]UNC-Chapel Hill

Abstract

We present HotStuff, a leader-based Byzantine fault-tolerant replication protocol for the partially synchronous model. Once network communication becomes synchronous, HotStuff enables a correct leader to drive the protocol to consensus at the pace of actual (vs. maximum) network delay—a property called *responsiveness*—and with communication complexity that is linear in the number of replicas. To our knowledge, HotStuff is the first partially synchronous BFT replication protocol exhibiting these combined properties. HotStuff is built around a novel framework that forms a bridge between classical BFT foundations and blockchains. It allows the expression of other known protocols (DLS, PBFT, Tendermint, Casper), and ours, in a common framework.

Our deployment of HotStuff over a network with over 100 replicas achieves throughput and latency comparable to that of BFT-SMaRt, while enjoying linear communication footprint during leader failover (vs. cubic with BFT-SMaRt).

1 Introduction

Byzantine fault tolerance (BFT) refers to the ability of a computing system to endure arbitrary (i.e., Byzantine) failures of its components while taking actions critical to the system's operation. In the context of state machine replication (SMR) [35, 47], the system as a whole provides a replicated service whose state is mirrored across n deterministic replicas. A BFT SMR protocol is used to ensure that non-faulty replicas agree on an order of execution for client-initiated service commands, despite the efforts of f Byzantine replicas. This, in turn, ensures that the $n-f$ non-faulty replicas will run commands identically and so produce the same response for each command. As is common, we are concerned here with the partially synchronous communication model [25], whereby a known bound Δ on message transmission holds after some unknown *global stabilization time* (GST). In this model, $n \geq 3f+1$ is required for non-faulty replicas to agree on the same commands in the same order (e.g., [12]) and progress can be ensured deterministically only after GST [27].

When BFT SMR protocols were originally conceived, a typical target system size was $n = 4$ or $n = 7$, deployed

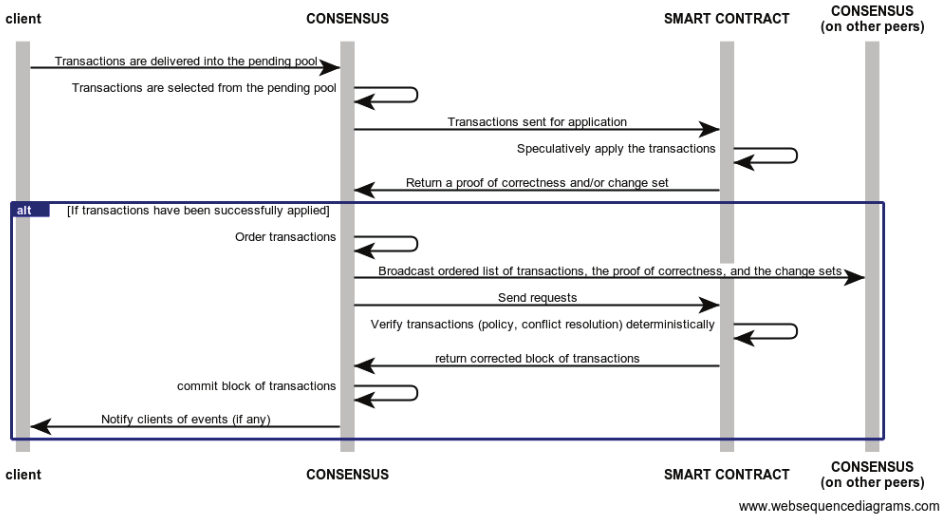
HotStuff

Algorithm 2 Basic HotStuff protocol (for replica r).

```
1:  for $curView \leftarrow 1, 2, 3, \ldots$ do
    ▷ PREPARE phase
2:    as a leader   // $r = \text{LEADER}(curView)$
      // we assume special NEW-VIEW messages from view 0
3:      wait for $(n - f)$ new-view messages: $M \leftarrow \{m \mid \text{matchingMsg}(m, \text{new-view}, curView - 1)\}$
4:      $highQC \leftarrow \left(\arg\max_{m \in M}\{m.justify.viewNumber\}\right).justify$
5:      $curProposal \leftarrow \text{createLeaf}(highQC.node, \text{client's command})$
6:      broadcast $\text{Msg}(\text{prepare}, curProposal, highQC)$
7:    as a replica
8:      wait for message $m$ : $\text{matchingMsg}(m, \text{prepare}, curView)$ from $\text{leader}(curView)$
9:      if $m.node$ extends from $m.justify.node \land$
                $\text{safeNode}(m.node, m.justify)$ then
10:       send $\text{voteMsg}(\text{prepare}, m.node, \perp)$ to $\text{leader}(curView)$
    ▷ PRE-COMMIT phase
11:   as a leader
12:     wait for $(n - f)$ votes: $V \leftarrow \{v \mid \text{matchingMsg}(v, \text{prepare}, curView)\}$
13:     $prepareQC \leftarrow \text{QC}(V)$
14:     broadcast $\text{Msg}(\text{pre-commit}, \perp, prepareQC)$
15:   as a replica
16:     wait for message $m$ : $\text{matchingQC}(m.justify, \text{prepare}, curView)$ from $\text{leader}(curView)$
17:     $prepareQC \leftarrow m.justify$
18:     send $\text{voteMsg}(\text{pre-commit}, m.justify.node, \perp)$ to $\text{leader}(curView)$
```

```
    ▷ COMMIT phase
19: as a leader
20:     wait for $(n - f)$ votes: $V \leftarrow \{v \mid \text{matchingMsg}(v, \text{pre-commit}, \textit{curView})\}$
21:     $\textit{precommitQC} \leftarrow \text{QC}(V)$
22:     broadcast $\text{Msg}(\text{commit}, \bot, \textit{precommitQC})$
23: as a replica
24:     wait for message $m$ : $\text{matchingQC}(m.\textit{justify}, \text{pre-commit}, \textit{curView})$ from $\text{leader}(\textit{curView})$
25:     $\textit{lockedQC} \leftarrow m.\textit{justify}$
26:     send $\text{voteMsg}(\text{commit}, m.\textit{justify}.\textit{node}, \bot)$ to $\text{leader}(\textit{curView})$
    ▷ DECIDE phase
27: as a leader
28:     wait for $(n - f)$ votes: $V \leftarrow \{v \mid \text{matchingMsg}(v, \text{commit}, \textit{curView})\}$
29:     $\textit{commitQC} \leftarrow \text{QC}(V)$
30:     broadcast $\text{Msg}(\text{decide}, \bot, \textit{commitQC})$
31: as a replica
32:     wait for message $m$ from $\text{leader}(\textit{curView})$
33:     wait for message $m$ : $\text{matchingQC}(m.\textit{justify}, \text{commit}, \textit{curView})$ from $\text{leader}(\textit{curView})$
34:     execute new commands through $m.\textit{justify}.\textit{node}$, respond to clients
    ▷ Finally
35: nextView interrupt: goto this line if $\text{nextView}(\textit{curView})$ is called during “wait for” in any phase
36: send $\text{Msg}(\text{new-view}, \bot, \textit{prepareQC})$ to $\text{leader}(\textit{curView} + 1)$
```

Hyperledger


client
CONSENSUS
SMART CONTRACT
CONSENSUS (on other peers)
Transactions are delivered into the pending pool
Transactions are selected from the pending pool
Transactions sent for application
Speculatively apply the transactions
Return a proof of correctness and/or change set
alt
[If transactions have been successfully applied]
Order transactions
Broadcast ordered list of transactions, the proof of correctness, and the change sets
Send requests
Verify transactions (policy, conflict resolution) deterministically
return corrected block of transactions
commit block of transactions
Notify clients of events (if any)
client
CONSENSUS
SMART CONTRACT
CONSENSUS (on other peers)
www.websequencediagrams.com

Hyperledger

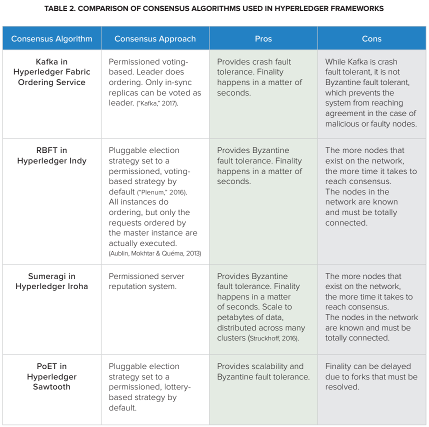
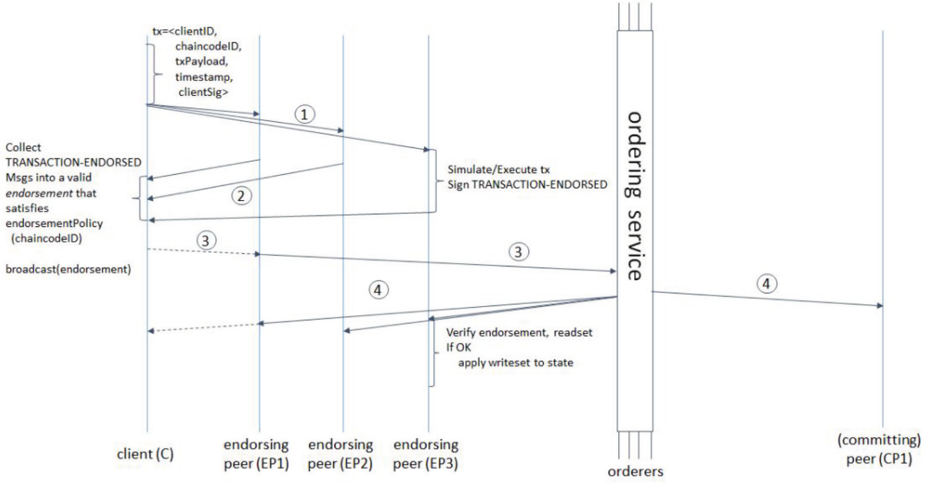


TABLE 2. COMPARISON OF CONSENSUS ALGORITHMS USED IN HYPERLEDGER FRAMEWORKS

Consensus Algorithm	Consensus Approach	Pros	Cons
Kafka in Hyperledger Fabric Ordering Service	Permissioned voting-based. Leader does ordering. Only in-sync replicas can be voted as leader. ("Kafka," 2017).	Provides crash fault tolerance. Finality happens in a matter of seconds.	While Kafka is crash fault tolerant, it is not Byzantine fault tolerant, which prevents the system from reaching agreement in the case of malicious or faulty nodes.
RBFT in Hyperledger Indy	Pluggable election strategy set to a permissioned, voting-based strategy by default ("Plenum," 2016). All instances do ordering, but only the requests ordered by the master instance are actually executed. (Aublin, Mokhtar & Quéma, 2013)	Provides Byzantine fault tolerance. Finality happens in a matter of seconds.	The more nodes that exist on the network, the more time it takes to reach consensus. The nodes in the network are known and must be totally connected.
Sumeragi in Hyperledger Iroha	Permissioned server reputation system.	Provides Byzantine fault tolerance. Finality happens in a matter of seconds. Scale to petabytes of data, distributed across many clusters (Struckhoff, 2016).	The more nodes that exist on the network, the more time it takes to reach consensus. The nodes in the network are known and must be totally connected.
PoET in Hyperledger Sawtooth	Pluggable election strategy set to a permissioned, lottery-based strategy by default.	Provides scalability and Byzantine fault tolerance.	Finality can be delayed due to forks that must be resolved.

Hyperledger



Virtual Machines

- Blockchains implement VMs for increased flexibility
- These VMs can be either
  1. non-Turing complete (bitcoin)
  2. Turing complete with bounds on resurce consumption (ethereum, libra, etc.)
- Blockchains as distributed computing frameworks;
- Blockchains as hosts for *smart contracts*.

Smart contracts

Procedures

- safely stored on a platform (blockchain)
- automatically triggered by events
- audited and controlled only by the platform itself (not server-side)
- able to trigger new events.

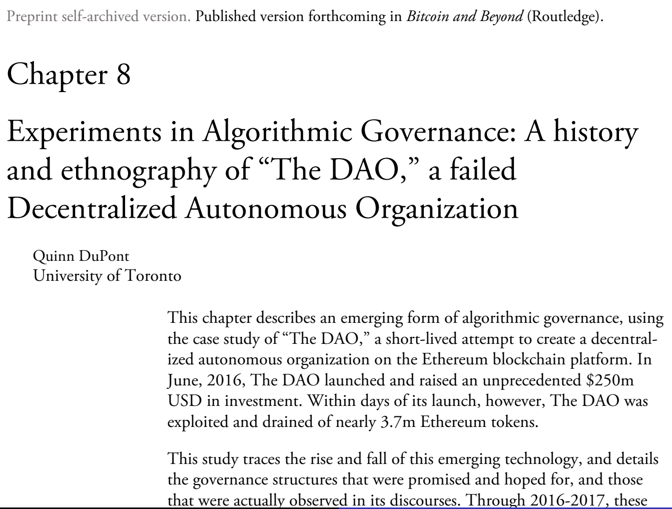
Remark

In general a smart contract acts *only* on the state of the blockchain.

Warning

Smart contracts can be dangerous and hard to debug.

Smart contracts



Chapter 8

Experiments in Algorithmic Governance: A history and ethnography of “The DAO,” a failed Decentralized Autonomous Organization

Quinn DuPont
University of Toronto

This chapter describes an emerging form of algorithmic governance, using the case study of “The DAO,” a short-lived attempt to create a decentralized autonomous organization on the Ethereum blockchain platform. In June, 2016, The DAO launched and raised an unprecedented $250m USD in investment. Within days of its launch, however, The DAO was exploited and drained of nearly 3.7m Ethereum tokens.

This study traces the rise and fall of this emerging technology, and details the governance structures that were promised and hoped for, and those that were actually observed in its discourses. Through 2016-2017, these

Applications to industry and robotics

- Auditability and tracking of manufacturing steps
- Robustness and replication of commands
- Distributed computation and collaborative logic
- Transparency and accountability
- Bidding and decentralized business models
- Economy of things (Machine to machine interaction)

M2M interaction in Industry 4.0

- On demand manufacturing
- Auditing and diagnostics
- Traceability
- Authentication
- Subscription production
- Quality and stock control

Division algorithm

Division algorithm

$\forall \alpha \in \mathbb{Z}, \beta \in \mathbb{Z} \setminus \{0\} : \exists q \in \mathbb{Z}, r \in \mathbb{N}$ such that

$$\alpha = \beta q + r, \qquad r < |\beta| \text{ or } r = 0$$

Definition

For any $\alpha \in \mathbb{Z}$, we say that $\gamma \in \mathbb{Z}$ divides α (in symbols $\gamma|\alpha$) if

$$\exists k \in \mathbb{Z} : \alpha = \gamma k.$$

GCD

Definition

For any $(\alpha, \beta) \in \mathbb{Z}^2 \setminus \{(0,0)\}$ we say that γ is a *greatest common divisor* between α and β if and only if

$$k|\alpha, \quad k|\beta$$

and for any t with $t|\alpha$ and $t|\beta$,

$$t|k.$$

Theorem

For any $\alpha, \beta \in \mathbb{Z}$, $(\alpha, \beta) \neq (0,0)$ there are exactly 2 greatest common divisors between them.

Diophantine equations

Theorem

For any $\alpha, \beta, \gamma \in \mathbb{Z}$*, the equation*

$$\alpha x + \beta y = \gamma \tag{1}$$

admits solutions $(\bar{x}, \bar{y}) \in \mathbb{Z}^2$ *if and only if* $\alpha = \beta = \gamma = 0$ *or* $(\alpha, \beta) \neq (0, 0)$ *and* $\gcd(\alpha, \beta) | \gamma$.

Theorem (Euclidean algorithm)

It is easy to compute $\gcd(\alpha, \beta)$ *and it is also easy to find the solutions of* (1).

Properties of GCDs

For all $\alpha, \beta \in \mathbb{Z}$:

- $\gcd(\alpha, \beta) = \gcd(\beta, \alpha)$;
- $\gcd(\alpha, 0) = \alpha$;
- if $\beta = \alpha q + r$, then $\gcd(\alpha, \beta) = \gcd(\alpha, r)$.

```
def Euclidean(u, v):
  if v > u:
      return Euclidean(v, u)
  while v != 0:
      q = u // v
      r = u - v * q
      u = v
      v = r
  return u
```

Extended Euclidean Algorithm

```
def XEuclidean(u, v):
    if v > u:
        return XEuclidean(v, u)
    U = numpy.array([1, 0, int(u)])
    V = numpy.array([0, 1, int(v)])
    while V[2] != 0:
        q = U[2] // V[2]
        R = U - V * q
        U = V
        V = R
    return U
```

Euclidean domains

Euclidean domain

- $(\mathbb{D}, +, \cdot)$: commutative ring with 1
- $\delta : \mathbb{D} \setminus \{0\} \to \mathbb{N}$ such that $\forall \alpha \in \mathbb{D}, \beta \in \mathbb{D} \setminus \{0\}$:
$$\exists q, r \in \mathbb{D} : \alpha = \beta q + r \text{ and } r = 0 \text{ or } \delta(r) < \delta(\beta).$$

Examples:

- $\mathbb{Z}$, $\delta(\cdot) := |\cdot|$;
- $\mathbb{F}$ field, $\delta(\cdot) := 1$;
- $\mathbb{F}[x]$ polynomial ring, $\delta(\cdot) := \deg(\cdot)$.

Remark

In Euclidean domains we can use the Euclidean algorithm.

- $(R, +, \cdot)$: commutative ring with 1;
- $I \trianglelefteq R$: ideal;
- $(R/I, +, \cdot)$: quotient ring with

$$(a+I)+(b+I)=(a+b)+I, \quad (a+I)(b+I)=(ab+I);$$

Remark

- $(R/I, +, \cdot)$: **integral domain** if and only if I prime;
- $(R/I, +, \cdot)$: **field** if and only if I maximal.

Proof. (maximal $\Leftrightarrow$ field).

$$\forall a \in R : a+I \neq I \Leftrightarrow a \in R \setminus I \Leftrightarrow$$
$$\overline{\{a\} \cup I} = R \Leftrightarrow 1 \in \overline{\{a\} \cup I} \Leftrightarrow \exists \alpha \in R, \beta \in I : a\alpha + \beta = 1$$
$$\Leftrightarrow (a+I)(\alpha+I) = 1+I$$

$\square$

Finite fields

Prime order fields

- p: prime;
- $p\mathbb{Z} := \{p\alpha : \alpha \in \mathbb{Z}\}$;
- $\mathbb{Z}_p := \mathbb{Z}/(p\mathbb{Z})$: field with p elements.

Proof.

Given $a \in \mathbb{Z}$, either $a \in p\mathbb{Z}$ or $\gcd(a, p) = 1$; by the extended Euclidean algorithm there exist $b, k \in \mathbb{Z}$ such that $ab + pk = 1$; thus

$$(a + p\mathbb{Z})(b + p\mathbb{Z}) = (ab + p\mathbb{Z}) = (1 - pk) + p\mathbb{Z} = 1 + p\mathbb{Z}.$$

$\square$

Finite fields

Prime power order fields

- p: prime; $n > 0$; $q = p^n$;
- $\mathbb{Z}_p$: finite field with p elements;
- $f(x) \in \mathbb{Z}_p[x]$ irreducible polynomial of degree n; $I = (f(x))$;
- $\mathbb{F}_q := \mathbb{Z}_p[x]/I$: field with q elements.

Proof.

Given $a(x) \in \mathbb{Z}[x]$, either $a(x) \in I$ or $\gcd(a(x), f(x)) = 1$; by the extended Euclidean algorithm there exist $b(x), k(x) \in \mathbb{Z}$ such that $a(x)b(x) + f(x)k(x) = 1$; thus

$$(a(x) + I)(b(x) + I) = (a(x)b(x) + I) = (1 - f(x)k(x)) + I = 1 + I.$$

□

Cyclic groups

- $(G,\cdot)$: finite cyclic group of order n;
- $(G,\cdot) \cong (\mathbb{Z}_n,+)$;
- $G=\langle g\rangle$; g generator of G;
- $G=\{g^\alpha : \alpha = 0,\dots,n-1\}$

Definition

For any $g \in G$, define $\mathrm{DLOG}_g : \langle g\rangle \to \{0,\dots,n-1\}$ as

$$\mathrm{DLOG}_g(x) := \beta \text{ such that } g^\beta = x, 0 \leq \beta \leq n-1.$$

Remarks

1. In $\mathbb{Z}_n$ solving DLOG_g is trivial.
2. It might be hard to compute an isomorphism $\xi : G \to \mathbb{Z}_n$ (DLP).

Finite fields: multiplicative group

Theorem

The group $(\mathbb{F}_q^*, \cdot)$ of all invertible elements of $\mathbb{F}_q$ is cyclic of order $q-1$.

Definition

- $\phi(d) := |\mathbb{Z}_d^*| = |\{a\colon 1 \leq a \leq q-1 : \gcd(a,d) = 1\}|$.
- $\psi_q(d) := |\{x \in \mathbb{F}_q^* : |\langle x \rangle| = d\}|$.

Finite fields: multiplicative group

Lemma

$$\forall n \in \mathbb{N} \setminus \{0\} : \sum_{d|n} \phi(d) = n.$$

Proof.

Any element $x \in \mathbb{Z}_n$ generates a cyclic subgroup C_d of $\mathbb{Z}_n$ of order d for some $d|n$; each subgroup C_d admits $\phi(d)$ generators. □

Lemma

$$\sum_{d|(q-1)} \psi_q(d) = q - 1 = \sum_{d|(q-1)} \phi(d) \qquad (2)$$

Proof.

Each element of $\mathbb{F}_q^*$ has some order $d|(q-1)$. □

Finite fields: multiplicative group

Theorem.

1. In a field $x^d - 1$ has at most d roots;
2. $C_d := \{x \in \mathbb{F}_q^* : x^d = 1\}$ has at most d elements; so $|C_d| \leq d$;
3. Let a be an element of order d with $d|(q-1)$; then $a \in C_d$; and $|C_d| \geq d$; so $|C_d| = d$, whence $C_d = \langle a \rangle$ is cyclic;
4. C_d has $\phi(d)$ distinct generators, all its elements of order d;
5. So either $\psi_q(d) = 0$ or $\psi_q(d) = |C_d| = \phi(d)$;
6. By (4), $\forall d|(q-1)$: $\psi_q(d) = \phi(d)$;
7. In particular, $\psi_q(q-1) = \phi(q-1) > 0$.

$\square$

Groups for cryptography

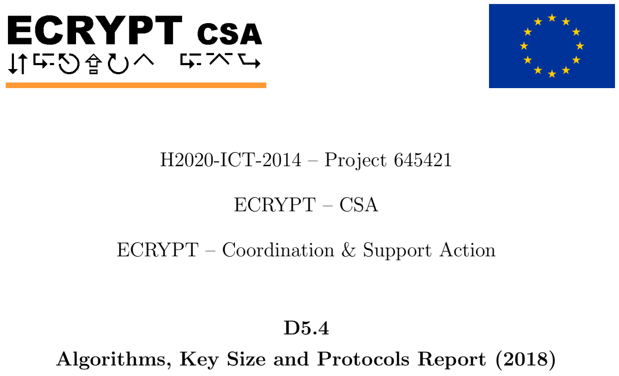
Algorithms based upon hardness of DLP

- (EC)DSA signature
- ElGamal encryption
- Diffie-Hellman key exchange

Good groups for DLP-based cryptography

- Multiplicative group of a finite field
- Group of the points of an elliptic curve

Recommended key sizes



ECRYPT CSA

H2020-ICT-2014 – Project 645421

ECRYPT – CSA

ECRYPT – Coordination & Support Action

D5.4

Algorithms, Key Size and Protocols Report (2018)

Recommended key sizes (2018)

	Legacy	Near Term	Long Term
Symmetric Key Size	80	128	256
RSA Problem	1024	3072	15360
Finite field DLP	1024	3072	15360
ECDLP	160	256	512

Elliptic curves

- $p > 3, q = p^n, a, b \in \mathbb{F}_q, 4a^3 + 27b^2 \neq 0$:

$$f(x,y) := y^2 - (x^3 + ax + b) \tag{3}$$

- $p = 2, q = 2^n, b \in \mathbb{F}_q, b \neq 0$:

$$f(x,y) := y^2 + xy - (x^3 + ax^2 + b) \tag{4}$$

- $\mathcal{E}(\mathbb{F}_q) := \{(x,y) \in \mathbb{F}_q^2 \colon f(x,y) = 0\} \cup \{\mathcal{O}_\infty := [(0,1,0)]\}$;
- $|\mathcal{E}(\mathbb{F}_q) - (q+1)| \leq 2\sqrt{q}$ (Hasse);
- $\mathcal{E}(\mathbb{F}_q)$ has $1, 3$ or 9 inflection points;
- $\mathcal{O}_\infty$ is an inflection.

Elliptic curves/remarks

Remark

A line ℓ intersecting $\mathcal{E}(\mathbb{F})$ in at least 2 points over $\mathbb{F}$ meets $\mathcal{E}(\mathbb{F})$ in exactly 3 points over $\mathbb{F}_q$.

Proof (informal sketch).

Let $\ell : y = ax + b$ and consider $g(x) := f(x, ax + b)$. Then, $\deg(g(x)) = 3$ and $(x - \zeta_1)|g(x)$, $(x - \zeta_2)|g(x)$ for elements $\zeta_1, \zeta_2 \in \mathbb{F}$; so $g(x) = (x - \zeta_1)(x - \zeta_2)(x - \gamma)$ splits in 3 linear factors and it has 3 roots. □

For $P, Q \in \mathcal{E}(\mathbb{F}_q)$, write $\ell(P, Q)$ for the line through P and Q and $\ell(P, P)$ for the tangent to $\mathcal{E}(\mathbb{F}_q)$ in P. The *intersection divisor* of $\ell(P, Q)$ and $\mathcal{E}(\mathbb{F}_q)$ is

$$\ell(P, Q).\mathcal{E}(\mathbb{F}_q) = P + Q + T.$$

Elliptic curves/group operation

Definition

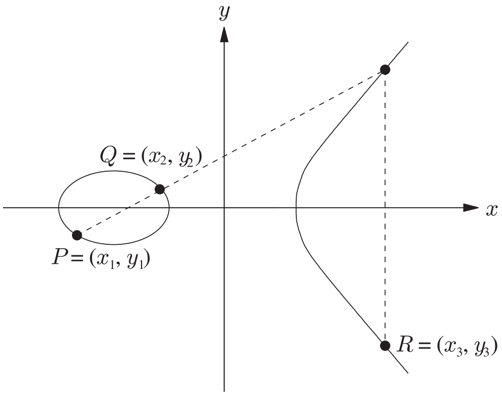
$$\ominus P := \ell(P, \mathcal{O}_\infty).\mathcal{E}(\mathbb{F}_q) - P - \mathcal{O}_\infty;$$

$$(P \oplus Q) := \ominus(\ell(P, Q).\mathcal{E}(\mathbb{F}_q) - (P + Q))$$

Theorem

$(\mathcal{E}(\mathbb{F}_q), \oplus)$ *is an Abelian group.*

Elliptic curves



Group law: $p > 3$

$$P \equiv (x_1, y_1), \quad Q \equiv (x_2, y_2), \quad R = P \oplus Q \equiv (x_3, y_3)$$

1. $P \oplus \mathcal{O}_\infty = \mathcal{O}_\infty \oplus P = P$;
2. $\ominus P \equiv (x_1, -y_1)$; $P \oplus (\ominus P) = \mathcal{O}_\infty$;
3. if $P \neq Q$, $P \neq \ominus Q$,
$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2, \quad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1;$$
4. if $P \neq \ominus P$ and $Q = P$, then $R = P \oplus P = 2P$,
$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1, \quad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

Group law/example

$$p = 23, \quad f(x,y) = y^2 - (x^3 + x + 4)$$

$(0,2)$	$(0,21)$	$(1,11)$	$(1,12)$	$(4,7)$	$(4,16)$
$(7,3)$	$(7,20)$	$(8,8)$	$(8,15)$	$(9,11)$	$(9,12)$
$(10,5)$	$(10,18)$	$(11,9)$	$(11,14)$	$(13,11)$	$(13,12)$
$(14,5)$	$(14,18)$	$(15,6)$	$(15,17)$	$(17,9)$	$(17,14)$
$(18,9)$	$(18,14)$	$(22,5)$	$(22,19)$	$\mathcal{O}_\infty$	

$$|\mathcal{E}(\mathbb{F}_{23})| = 29 < 24 + 2\sqrt{23} = 32$$

$$\mathcal{E}(\mathbb{F}_{23}) = \langle (0,2) \rangle.$$

- $P \equiv (4,7), \quad Q \equiv (13,11)$
- $\ominus P \equiv (4,-7) = (4,16)$
- $P \oplus Q \equiv (15,6)$
- $2P = (10,18)$

Group law: $p = 2$

$$P \equiv (x_1, y_1), \quad Q \equiv (x_2, y_2), \quad R = P \oplus Q \equiv (x_3, y_3)$$

1. $P \oplus \mathcal{O}_\infty = \mathcal{O}_\infty \oplus P = P$;
2. $\ominus P \equiv (x_1, x_1 + y_1); P \oplus (\ominus P) = \mathcal{O}_\infty$;
3. if $P \neq Q, \ominus Q$

$$x_3 = \left(\frac{y_2 + y_1}{x_2 + x_1}\right)^2 + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a, \quad y_3 = \left(\frac{y_2 + y_1}{x_2 + x_1}\right)(x_1 + x_3) + x_3 + y_1;$$

4. if $P \neq \ominus P$ and $Q = P$, then $R = P \oplus P = 2P$,

$$x_3 = x_1^2 + \frac{b}{x_1^2}, \quad y_3 = x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right) x_3 + x_3.$$

Group structure

Theorem

Let $n = |\mathcal{E}(\mathbb{F}_q)|$*; then,* $\mathcal{E}(\mathbb{F}_q)$ *with the operations introduced above is an abelian group and either* $\mathcal{E}(\mathbb{F}_q) \cong \mathbb{Z}_n$ *or*

$$\mathcal{E}(\mathbb{F}_q) \cong \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$$

with $n = n_1 n_2$ *and* $n_2 | n_1$.

Hidden subgroup problem (HSP)

Let G be a group, $H \leq G$ be a subgroup and X be a set.

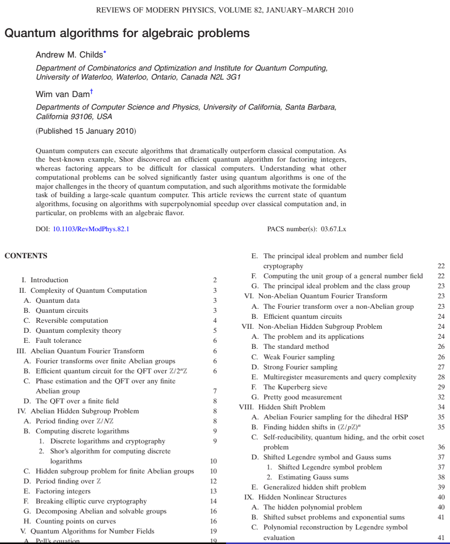
Definition

- A function $f : G \to X$ such that $f(g_1) = f(g_2)$ if and only if $g_1^{-1} g_2 \in H$ *hides* H.
- Given G, X the HSP consists in recovering a generating set of H using $O(\log|G| + \log|X|)$ evaluations of f.

Applications

- $f_x : \begin{cases} \mathbb{Z}_N \times \mathbb{Z}_N \to G = \langle g \rangle \\ (\alpha, \beta) \to x^\alpha g^\beta \end{cases}$ hides $H = \{(\alpha, \alpha\,\mathrm{DLOG}_g\, x) : \alpha \in \mathbb{Z}_N\}$
- $f_x : \begin{cases} \mathbb{Z} \to \mathbb{Z}_N \\ \alpha \to x^\alpha \pmod N \end{cases}$ hides $H = \{r : x^r = 1\}$

Hidden subgroup problem/QC



REVIEWS OF MODERN PHYSICS, VOLUME 82, JANUARY–MARCH 2010

Quantum algorithms for algebraic problems

Andrew M. Childs[*]

Department of Combinatorics and Optimization and Institute for Quantum Computing, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Wim van Dam[†]

Departments of Computer Science and Physics, University of California, Santa Barbara, California 93106, USA

(Published 15 January 2010)

Quantum computers can execute algorithms that dramatically outperform classical computation. As the best-known example, Shor discovered an efficient quantum algorithm for factoring integers, whereas factoring appears to be difficult for classical computers. Understanding what other computational problems can be solved significantly faster using quantum algorithms is one of the major challenges in the theory of quantum computation, and such algorithms motivate the formidable task of building a large-scale quantum computer. This article reviews the current state of quantum algorithms, focusing on algorithms with superpolynomial speedup over classical computation and, in particular, on problems with an algebraic flavor.

DOI: 10.1103/RevModPhys.82.1 PACS number(s): 03.67.Lx

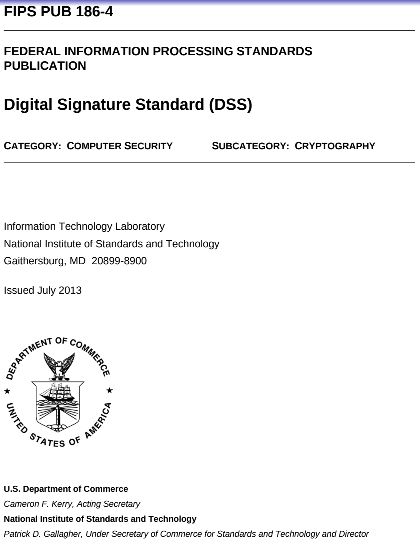
CONTENTS

Groups for cryptography

Remarks

- Quantum Computers promise to break DLP.
- As of today (2020), the groups $(\mathbb{F}_q^*, \cdot)$ are considered the benchmark for all algorithms based on DLP.
- The groups $\mathcal{E}(\mathbb{F}_q)$ are comparatively much more secure than $\mathbb{F}_q^*$.
- Different approaches might be needed in the long term.

DSA



FIPS PUB 186-4

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

Digital Signature Standard (DSS)

CATEGORY: COMPUTER SECURITY **SUBCATEGORY: CRYPTOGRAPHY**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

Issued July 2013

DEPARTMENT OF COMMERCE ★ UNITED STATES OF AMERICA ★

U.S. Department of Commerce
Cameron F. Kerry, Acting Secretary
National Institute of Standards and Technology
Patrick D. Gallagher, Under Secretary of Commerce for Standards and Technology and Director

DSA

- *Domain parameters generation*
  1. p: prime, q: prime, $q|(p-1)$.
  2. g: generator of the unique cyclic subgroup of order q in $\mathbb{F}_p^*$
  3. return p,q and g.
- *Key Generation*
  1. $x \leftarrow \mathrm{Random}(1; q-1)$
  2. $y \leftarrow g^x \pmod{p}$
  3. y: public key; x: private key

DSA

- *Sign a message m*
  1. $k \leftarrow \mathrm{Nonce}(1; q-1)$
  2. $X \leftarrow g^k \pmod{p}; r \leftarrow X \pmod{q}$
  3. if $r = 0$, then return to 1
  4. use the euclidean algorithm to compute $k^{-1} \pmod{q}$
  5. $s \leftarrow k^{-1}(m + xr) \pmod{q}$
  6. if $s = 0$, then return to 1
  7. return (r, s)
- *Verify Signature given* (m, r, s)
  1. $w \leftarrow s^{-1} \pmod{q}$
  2. $u_1 \leftarrow mw \pmod{q}; u_2 \leftarrow rw \pmod{q}$
  3. $X \leftarrow g^{u_1} y^{u_2} \pmod{p}; v \leftarrow X \pmod{q}$
  4. Verify if $v = r$.

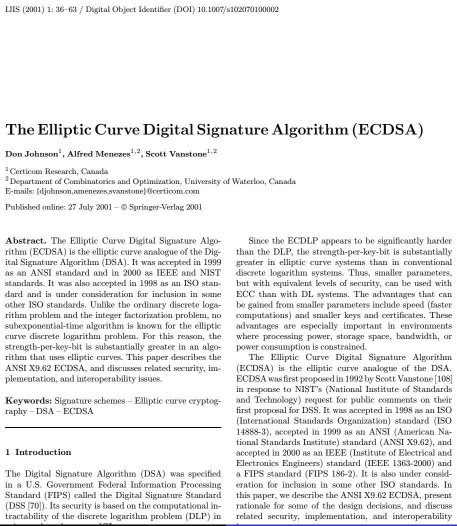
DSA

Proof.

$$v = X \pmod{q} = g^{u_1} y^{u_2} = g^{mw} y^{rw} = g^{s^{-1}m} y^{s^{-1}r} = g^{k(m+xr)^{-1}m} y^{k(m+xr)^{-1}r} = g^{k(m+xr)^{-1}m} g^{k(m+xr)^{-1}rx} = g^{k(m+xr)^{-1}(m+rx)} = g^k = r$$

□

ECDSA



IJIS (2001) 1: 36–63 / Digital Object Identifier (DOI) 10.1007/s102070100002

The Elliptic Curve Digital Signature Algorithm (ECDSA)

Don Johnson[1], **Alfred Menezes**[1,2], **Scott Vanstone**[1,2]

[1] Certicom Research, Canada
[2] Department of Combinatorics and Optimization, University of Waterloo, Canada
E-mails: {djohnson,amenezes,svanstone}@certicom.com

Published online: 27 July 2001 – © Springer-Verlag 2001

Abstract. The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA). It was accepted in 1999 as an ANSI standard and in 2000 as IEEE and NIST standards. It was also accepted in 1998 as an ISO standard and is under consideration for inclusion in some other ISO standards. Unlike the ordinary discrete logarithm problem and the integer factorization problem, no subexponential-time algorithm is known for the elliptic curve discrete logarithm problem. For this reason, the strength-per-key-bit is substantially greater in an algorithm that uses elliptic curves. This paper describes the ANSI X9.62 ECDSA, and discusses related security, implementation, and interoperability issues.

Keywords: Signature schemes – Elliptic curve cryptography – DSA – ECDSA

1 Introduction

The Digital Signature Algorithm (DSA) was specified in a U.S. Government Federal Information Processing Standard (FIPS) called the Digital Signature Standard (DSS [70]). Its security is based on the computational intractability of the discrete logarithm problem (DLP) in

Since the ECDLP appears to be significantly harder than the DLP, the strength-per-key-bit is substantially greater in elliptic curve systems than in conventional discrete logarithm systems. Thus, smaller parameters, but with equivalent levels of security, can be used with ECC than with DL systems. The advantages that can be gained from smaller parameters include speed (faster computations) and smaller keys and certificates. These advantages are especially important in environments where processing power, storage space, bandwidth, or power consumption is constrained.

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the DSA. ECDSA was first proposed in 1992 by Scott Vanstone [108] in response to NIST's (National Institute of Standards and Technology) request for public comments on their first proposal for DSS. It was accepted in 1998 as an ISO (International Standards Organization) standard (ISO 14888-3), accepted in 1999 as an ANSI (American National Standards Institute) standard (ANSI X9.62), and accepted in 2000 as an IEEE (Institute of Electrical and Electronics Engineers) standard (IEEE 1363-2000) and a FIPS standard (FIPS 186-2). It is also under consideration for inclusion in some other ISO standards. In this paper, we describe the ANSI X9.62 ECDSA, present rationale for some of the design decisions, and discuss related security, implementation, and interoperability

ECDSA

- *Domain parameters generation*
  1. q: prime power;
  2. $a, b \leftarrow \mathrm{Random}(\mathbb{F}_q)$;
  3. $N \leftarrow |\mathcal{E}(\mathbb{F}_q)|$;
  4. Check N divisible by a large prime n; else return to 2;
  5. Check N does not divide $q^k - 1$ for $1 \leq k \leq 20$; else return to 2;
  6. Check $n \neq q$; else return to 2;
  7. $G' \leftarrow \mathrm{Random}(\mathcal{E}(\mathbb{F}_q))$;
  8. $G \leftarrow (N/n)G'$; if $G = \mathcal{O}_\infty$ return to 7;
  9. return q, a, b, n, G.
- *Key Generation*
  1. $d \leftarrow \mathrm{Random}(1; n-1)$
  2. $Q \leftarrow dG$
  3. Q: public key; d: private key

ECDSA

- *Sign a message m*
  1. $k \leftarrow \mathrm{Nonce}(1; n-1)$
  2. $H \equiv (x_1, y_1) \leftarrow kG$; represent x_1 as an integer $\overline{x_1}$; $r \leftarrow \overline{x_1} \pmod{n}$;
  3. if $r = 0$, then return to 1;
  4. use the euclidean algorithm to compute $k^{-1} \pmod{n}$
  5. $s \leftarrow k^{-1}(m + dr) \pmod{n}$
  6. if $s = 0$, then return to 1
  7. return (r, s)
- *Verify Signature given* (m, r, s)
  1. $w \leftarrow s^{-1} \pmod{n}$;
  2. $u_1 \leftarrow mw \pmod{n}$; $u_2 \leftarrow rw \pmod{n}$;
  3. $X \equiv (x_1, y_1) \leftarrow u_1 G \oplus u_2 Q$;
  4. If $X = \mathcal{O}_\infty$, then reject;
  5. $v \leftarrow \overline{x_1}$;
  6. Verify if $v = r$.

ECDSA

Proof.

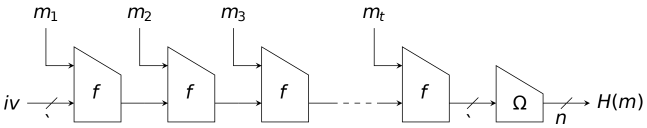
$$k = s^{-1}(m + dr) = w(m + rd) = u_1 + u_2 d \pmod{n},$$

whence

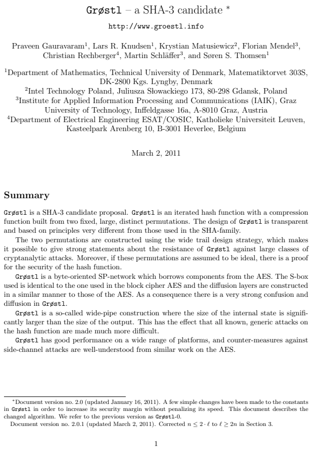
$$u_1 G \oplus u_2 Q = (u_1 + u_2 d)G = kG$$

Thus $v = r$. □

Merkle–Damgård



Grøstl



Grøstl – a SHA-3 candidate [*]

http://www.groestl.info

Praveen Gauravaram[1], Lars R. Knudsen[1], Krystian Matusiewicz[2], Florian Mendel[3], Christian Rechberger[4], Martin Schläffer[3], and Søren S. Thomsen[1]

[1]Department of Mathematics, Technical University of Denmark, Matematiktorvet 303S, DK-2800 Kgs. Lyngby, Denmark
[2]Intel Technology Poland, Juliusza Słowackiego 173, 80-298 Gdansk, Poland
[3]Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
[4]Department of Electrical Engineering ESAT/COSIC, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium

March 2, 2011

Summary

Grøstl is a SHA-3 candidate proposal. Grøstl is an iterated hash function with a compression function built from two fixed, large, distinct permutations. The design of Grøstl is transparent and based on principles very different from those used in the SHA-family.

The two permutations are constructed using the wide trail design strategy, which makes it possible to give strong statements about the resistance of Grøstl against large classes of cryptanalytic attacks. Moreover, if these permutations are assumed to be ideal, there is a proof for the security of the hash function.

Grøstl is a byte-oriented SP-network which borrows components from the AES. The S-box used is identical to the one used in the block cipher AES and the diffusion layers are constructed in a similar manner to those of the AES. As a consequence there is a very strong confusion and diffusion in Grøstl.

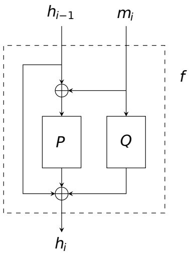
Grøstl is a so-called wide-pipe construction where the size of the internal state is significantly larger than the size of the output. This has the effect that all known, generic attacks on the hash function are made much more difficult.

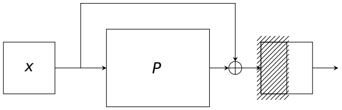
Grøstl has good performance on a wide range of platforms, and counter-measures against side-channel attacks are well-understood from similar work on the AES.

[*]Document version no. 2.0 (updated January 16, 2011). A few simple changes have been made to the constants in Grøstl in order to increase its security margin without penalizing its speed. This document describes the changed algorithm. We refer to the previous version as Grøstl-0.
Document version no. 2.0.1 (updated March 2, 2011). Corrected $n \leq 2 \cdot \ell$ to $\ell \geq 2n$ in Section 3.

1

Grøstl



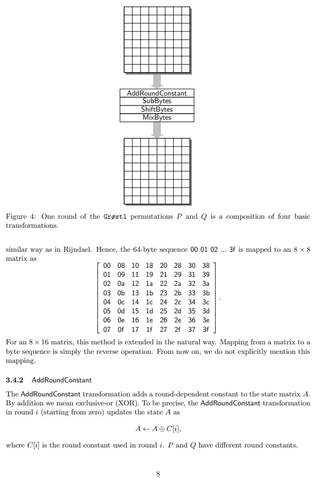


Grøstl/AES S-Boxes

- $m(x) = x^8 + x^4 + x^3 + x + 1$
- $\mathbb{F}_{256} = \mathbb{Z}_2[x]/(m(x))$
- Each element $\mathbf{b}$ of $\mathbb{F}_{256}$ reads as $(b_0, \ldots, b_7) \in \mathbb{Z}_2^8$
-

$$\mathrm{SubBytes}(\mathbf{b}) := \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \mathbf{b}^{254} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Grøstl



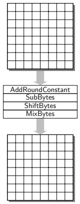


Figure 4: One round of the Grøstl permutations P and Q is a composition of four basic transformations.

similar way as in Rijndael. Hence, the 64-byte sequence 00 01 02 ... 3f is mapped to an 8×8 matrix as

$$\begin{bmatrix} 00 & 08 & 10 & 18 & 20 & 28 & 30 & 38 \\ 01 & 09 & 11 & 19 & 21 & 29 & 31 & 39 \\ 02 & 0a & 12 & 1a & 22 & 2a & 32 & 3a \\ 03 & 0b & 13 & 1b & 23 & 2b & 33 & 3b \\ 04 & 0c & 14 & 1c & 24 & 2c & 34 & 3c \\ 05 & 0d & 15 & 1d & 25 & 2d & 35 & 3d \\ 06 & 0e & 16 & 1e & 26 & 2e & 36 & 3e \\ 07 & 0f & 17 & 1f & 27 & 2f & 37 & 3f \end{bmatrix}.$$

For an 8×16 matrix, this method is extended in the natural way. Mapping from a matrix to a byte sequence is simply the reverse operation. From now on, we do not explicitly mention this mapping.

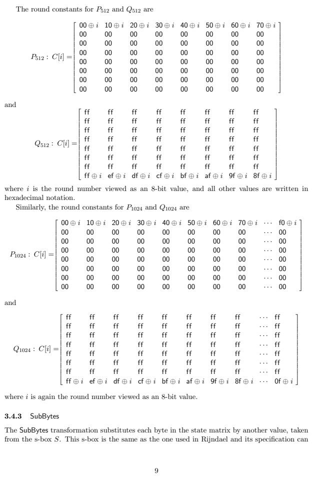
3.4.2 AddRoundConstant

The AddRoundConstant transformation adds a round-dependent constant to the state matrix A. By addition we mean exclusive-or (XOR). To be precise, the AddRoundConstant transformation in round i (starting from zero) updates the state A as

$$A \leftarrow A \oplus C[i],$$

where $C[i]$ is the round constant used in round i. P and Q have different round constants.

8



The round constants for P_{512} and Q_{512} are

$$P_{512}: C[i] = \begin{bmatrix} 00 \oplus i & 10 \oplus i & 20 \oplus i & 30 \oplus i & 40 \oplus i & 50 \oplus i & 60 \oplus i & 70 \oplus i \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{bmatrix}$$

and

$$Q_{512}: C[i] = \begin{bmatrix} ff & ff & ff & ff & ff & ff & ff & ff \\ ff & ff & ff & ff & ff & ff & ff & ff \\ ff & ff & ff & ff & ff & ff & ff & ff \\ ff & ff & ff & ff & ff & ff & ff & ff \\ ff & ff & ff & ff & ff & ff & ff & ff \\ ff & ff & ff & ff & ff & ff & ff & ff \\ ff & ff & ff & ff & ff & ff & ff & ff \\ ff \oplus i & ef \oplus i & df \oplus i & cf \oplus i & bf \oplus i & af \oplus i & 9f \oplus i & 8f \oplus i \end{bmatrix}$$

where i is the round number viewed as an 8-bit value, and all other values are written in hexadecimal notation.

Similarly, the round constants for P_{1024} and Q_{1024} are

$$P_{1024}: C[i] = \begin{bmatrix} 00 \oplus i & 10 \oplus i & 20 \oplus i & 30 \oplus i & 40 \oplus i & 50 \oplus i & 60 \oplus i & 70 \oplus i & \cdots & f0 \oplus i \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \cdots & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \cdots & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \cdots & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \cdots & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \cdots & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \cdots & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \cdots & 00 \end{bmatrix}$$

and

$$Q_{1024}: C[i] = \begin{bmatrix} ff & ff & ff & ff & ff & ff & ff & ff & \cdots & ff \\ ff & ff & ff & ff & ff & ff & ff & ff & \cdots & ff \\ ff & ff & ff & ff & ff & ff & ff & ff & \cdots & ff \\ ff & ff & ff & ff & ff & ff & ff & ff & \cdots & ff \\ ff & ff & ff & ff & ff & ff & ff & ff & \cdots & ff \\ ff & ff & ff & ff & ff & ff & ff & ff & \cdots & ff \\ ff & ff & ff & ff & ff & ff & ff & ff & \cdots & ff \\ ff \oplus i & ef \oplus i & df \oplus i & cf \oplus i & bf \oplus i & af \oplus i & 9f \oplus i & 8f \oplus i & \cdots & 0f \oplus i \end{bmatrix}$$

where i is again the round number viewed as an 8-bit value.

3.4.3 SubBytes

The SubBytes transformation substitutes each byte in the state matrix by another value, taken from the s-box S. This s-box is the same as the one used in Rijndael and its specification can

9

Grøstl

be found in Appendix B. Hence, if $a_{i,j}$ is the element in row i and column j of A, then SubBytes performs the following transformation:

$$a_{i,j} \leftarrow S(a_{i,j}), \quad 0 \le i < 8, \; 0 \le j < v.$$

See Figure 5.

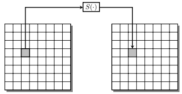


Figure 5: SubBytes substitutes each byte of the state by its image under the s-box S.

3.4.4 ShiftBytes and ShiftBytesWide

ShiftBytes and ShiftBytesWide cyclically shift the bytes within a row to the left by a number of positions. Let $\sigma = [\sigma_0, \sigma_1, \ldots, \sigma_7]$ be a list of distinct integers in the range from 0 to $v-1$. Then, ShiftBytes moves all bytes in row i of the state matrix σ_i positions to the left, wrapping around as necessary. The vector σ in ShiftBytes respectively ShiftBytesWide is different for P and Q. For ShiftBytes in P, we use $\sigma = [0,1,2,3,4,5,6,7]$ and for ShiftBytes in Q, we use $\sigma = [1,3,5,7,0,2,4,6]$. Similarly, for ShiftBytesWide in P and Q, we use $\sigma = [0,1,2,3,4,5,6,11]$ and $\sigma = [1,3,5,11,0,2,4,6]$ respectively. The transformations ShiftBytes and ShiftBytesWide for P and Q are illustrated in Figure 6 and Figure 7.

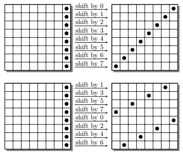


Figure 6: The ShiftBytes transformation of permutation P_{512} (top) and Q_{512} (bottom).

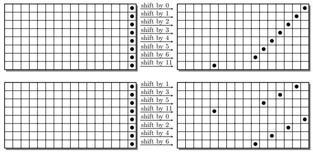


Figure 7: The ShiftBytesWide transformation of permutation P_{1024} (top) and Q_{1024} (bottom).

3.4.5 MixBytes

In the MixBytes transformation, each column in the matrix is transformed independently. To describe this transformation we first need to introduce the finite field $\mathbb{F}_{256}$. This finite field is defined in the same way as in Rijndael via the irreducible polynomial $x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$ over $\mathbb{F}_2$. The bytes of the state matrix A can be seen as elements of $\mathbb{F}_{256}$, i.e., as polynomials of degree at most 7 with coefficients in $\{0,1\}$. The least significant bit of each byte determines the coefficient of x^0, etc.

MixBytes multiplies each column of A by a constant 8×8 matrix B in $\mathbb{F}_{256}$. Hence, the transformation on the whole matrix A can be written as the matrix multiplication

$$A \leftarrow B \times A.$$

The matrix B is specified as

$$B = \begin{bmatrix} 02 & 02 & 03 & 04 & 05 & 03 & 05 & 07 \\ 07 & 02 & 02 & 03 & 04 & 05 & 03 & 05 \\ 05 & 07 & 02 & 02 & 03 & 04 & 05 & 03 \\ 03 & 05 & 07 & 02 & 02 & 03 & 04 & 05 \\ 05 & 03 & 05 & 07 & 02 & 02 & 03 & 04 \\ 04 & 05 & 03 & 05 & 07 & 02 & 02 & 03 \\ 03 & 04 & 05 & 03 & 05 & 07 & 02 & 02 \\ 02 & 03 & 04 & 05 & 03 & 05 & 07 & 02 \end{bmatrix}.$$

This matrix is *circulant*, which means that each row is equal to the row above rotated right by one position. In short, we may write $B = \mathrm{circ}(02, 02, 03, 04, 05, 03, 05, 07)$ instead. See also Figure 8.

Grøstl

be found in Appendix B. Hence, if $a_{i,j}$ is the element in row i and column j of A, then SubBytes performs the following transformation:

$$a_{i,j} \leftarrow S(a_{i,j}), \quad 0 \le i < 8, \ 0 \le j < v.$$

See Figure 5.

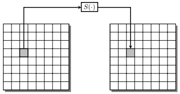


Figure 5: SubBytes substitutes each byte of the state by its image under the s-box S.

3.4.4 ShiftBytes and ShiftBytesWide

ShiftBytes and ShiftBytesWide cyclically shift the bytes within a row to the left by a number of positions. Let $\sigma = [\sigma_0, \sigma_1, \dots, \sigma_7]$ be a list of distinct integers in the range from 0 to $v-1$. Then, ShiftBytes moves all bytes in row i of the state matrix σ_i positions to the left, wrapping around as necessary. The vector σ in ShiftBytes respectively ShiftBytesWide is different for P and Q. For ShiftBytes in P, we use $\sigma = [0,1,2,3,4,5,6,7]$ and for ShiftBytes in Q, we use $\sigma = [1,3,5,7,0,2,4,6]$. Similarly, for ShiftBytesWide in P and Q, we use $\sigma = [0,1,2,3,4,5,6,11]$ and $\sigma = [1,3,5,11,0,2,4,6]$ respectively. The transformations ShiftBytes and ShiftBytesWide for P and Q are illustrated in Figure 6 and Figure 7.

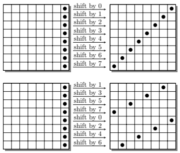


Figure 6: The ShiftBytes transformation of permutation P_{512} (top) and Q_{512} (bottom).

10

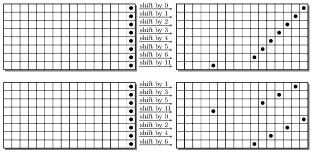


Figure 7: The ShiftBytesWide transformation of permutation P_{1024} (top) and Q_{1024} (bottom).

3.4.5 MixBytes

In the MixBytes transformation, each column in the matrix is transformed independently. To describe this transformation we first need to introduce the finite field $\mathbb{F}_{256}$. This finite field is defined in the same way as in Rijndael via the irreducible polynomial $x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$ over $\mathbb{F}_2$. The bytes of the state matrix A can be seen as elements of $\mathbb{F}_{256}$, i.e., as polynomials of degree at most 7 with coefficients in $\{0,1\}$. The least significant bit of each byte determines the coefficient of x^0, etc.

MixBytes multiplies each column of A by a constant 8×8 matrix B in $\mathbb{F}_{256}$. Hence, the transformation on the whole matrix A can be written as the matrix multiplication

$$A \leftarrow B \times A.$$

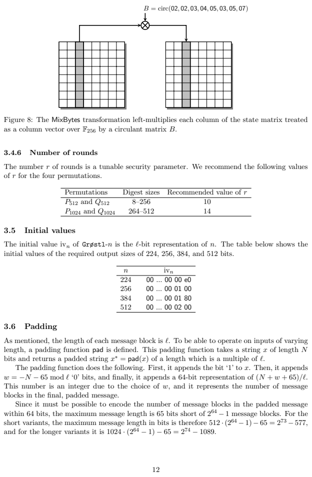
The matrix B is specified as

$$B = \begin{bmatrix} 02 & 02 & 03 & 04 & 05 & 03 & 05 & 07 \\ 07 & 02 & 02 & 03 & 04 & 05 & 03 & 05 \\ 05 & 07 & 02 & 02 & 03 & 04 & 05 & 03 \\ 03 & 05 & 07 & 02 & 02 & 03 & 04 & 05 \\ 05 & 03 & 05 & 07 & 02 & 02 & 03 & 04 \\ 04 & 05 & 03 & 05 & 07 & 02 & 02 & 03 \\ 03 & 04 & 05 & 03 & 05 & 07 & 02 & 02 \\ 02 & 03 & 04 & 05 & 03 & 05 & 07 & 02 \end{bmatrix}.$$

This matrix is *circulant*, which means that each row is equal to the row above rotated right by one position. In short, we may write $B = \mathrm{circ}(02, 02, 03, 04, 05, 03, 05, 07)$ instead. See also Figure 8.

11

Grøstl



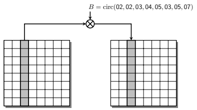


Figure 8: The MixBytes transformation left-multiplies each column of the state matrix treated as a column vector over $\mathbb{F}_{256}$ by a circulant matrix B.

3.4.6 Number of rounds

The number r of rounds is a tunable security parameter. We recommend the following values of r for the four permutations.

Permutations	Digest sizes	Recommended value of r
P_{512} and Q_{512}	8–256	10
P_{1024} and Q_{1024}	264–512	14

3.5 Initial values

The initial value iv_n of Grøstl-n is the ℓ-bit representation of n. The table below shows the initial values of the required output sizes of 224, 256, 384, and 512 bits.

n	iv_n
224	00 ... 00 00 e0
256	00 ... 00 01 00
384	00 ... 00 01 80
512	00 ... 00 02 00

3.6 Padding

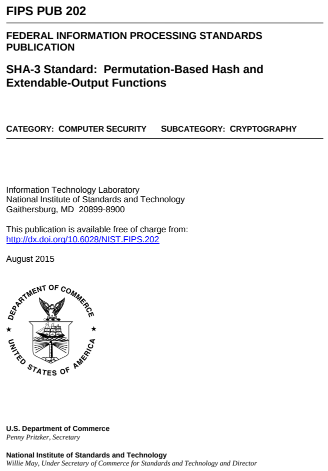
As mentioned, the length of each message block is ℓ. To be able to operate on inputs of varying length, a padding function pad is defined. This padding function takes a string x of length N bits and returns a padded string $x^* = \mathsf{pad}(x)$ of a length which is a multiple of ℓ.

The padding function does the following. First, it appends the bit '1' to x. Then, it appends $w = -N - 65 \bmod \ell$ '0' bits, and finally, it appends a 64-bit representation of $(N + w + 65)/\ell$. This number is an integer due to the choice of w, and it represents the number of message blocks in the final, padded message.

Since it must be possible to encode the number of message blocks in the padded message within 64 bits, the maximum message length is 65 bits short of $2^{64} - 1$ message blocks. For the short variants, the maximum message length in bits is therefore $512 \cdot (2^{64} - 1) - 65 = 2^{73} - 577$, and for the longer variants it is $1024 \cdot (2^{64} - 1) - 65 = 2^{74} - 1089$.

12

Keccak/SHA-3



FIPS PUB 202

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions

CATEGORY: COMPUTER SECURITY SUBCATEGORY: CRYPTOGRAPHY

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.FIPS.202

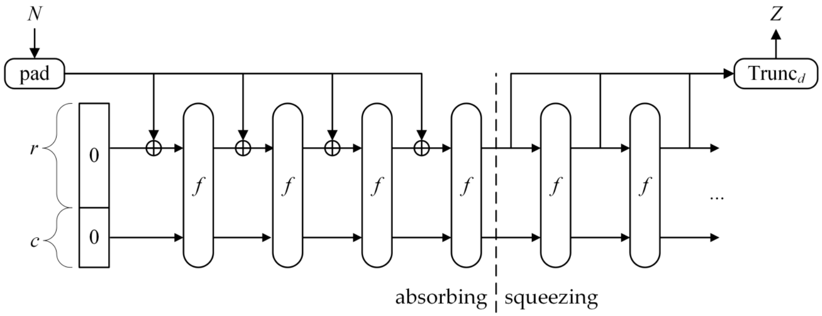
August 2015

DEPARTMENT OF COMMERCE ★ UNITED STATES OF AMERICA ★

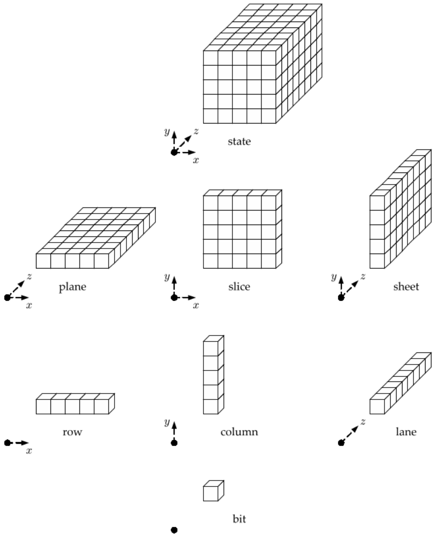
U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

Sponge construction



State array



Operation

3.1.4 Labeling Convention for the State Array

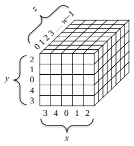


Figure 2: The *x*, *y*, and *z* coordinates for the diagrams of the step mappings

In the diagrams of the state that accompany the specifications of the step mappings, the lane that corresponds to the coordinates $(x, y) = (0, 0)$ is depicted at the center of the slices. The complete labeling of the x, y, and z coordinates for those diagrams is shown in Figure 2 above.

3.2 Step Mappings

The five step mappings that comprise a round of KECCAK-$p[b, n_r]$ are denoted by θ, ρ, π, χ, and ι. Specifications for these functions are given in Secs. 3.2.1-3.2.5.

The algorithm for each step mapping takes a state array, denoted by **A**, as an input and returns an updated state array, denoted by $\mathbf{A'}$, as the output. The size of the state is a parameter that is omitted from the notation, because b is always specified when the step mappings are invoked.

The ι mapping i_r has a second input: an integer called the *round index*, denoted by i_r, which is defined within Algorithm 7 for KECCAK-$p[b, n_r]$, in Sec. 3.3. The other step mappings do not depend on the round index.

3.2.1 Specification of θ

Algorithm 1: $\theta(\mathbf{A})$

Input:
state array **A**.

Output:
state array $\mathbf{A'}$.

11

Steps:
1. For all pairs (x, z) such that $0 \le x < 5$ and $0 \le z < w$, let
   $C[x, z] = \mathbf{A}[x, 0, z] \oplus \mathbf{A}[x, 1, z] \oplus \mathbf{A}[x, 2, z] \oplus \mathbf{A}[x, 3, z] \oplus \mathbf{A}[x, 4, z]$.
2. For all pairs (x, z) such that $0 \le x < 5$ and $0 \le z < w$ let
   $D[x, z] = C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z-1) \bmod w]$.
3. For all triples (x, y, z) such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let
   $\mathbf{A'}[x, y, z] = \mathbf{A}[x, y, z] \oplus D[x, z]$.

The effect of θ is to XOR each bit in the state with the parities of two columns in the array. In particular, for the bit $\mathbf{A}[x_0, y_0, z_0]$, the x-coordinate of one of the columns is $(x_0 - 1) \bmod 5$, with the same z-coordinate, z_0, while the x-coordinate of the other column is $(x_0 + 1) \bmod 5$, with z-coordinate $(z_0 - 1) \bmod w$.

In the illustration of the θ step mapping in Figure 3 below, the summation symbol, $\sum$, indicates the parity, i.e., the XOR sum of all the bits in the column.

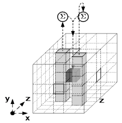


Figure 3: Illustration of θ applied to a single bit [8]

3.2.2 Specification of ρ

Algorithm 2: $\rho(\mathbf{A})$

Input:
state array **A**.

Output:
state array $\mathbf{A'}$.

Steps:
1. For all z such that $0 \le z < w$, let $\mathbf{A'}[0, 0, z] = \mathbf{A}[0, 0, z]$.
2. Let $(x, y) = (1, 0)$.

12

Operation

```
3. For $t$ from 0 to 23:
    a. for all $z$ such that $0 \le z < w$, let $\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, (z-(t+1)(t+2)/2) \bmod w]$;
    b. let $(x, y) = (y, (2x+3y) \bmod 5)$.
4. Return $\mathbf{A}'$.
```

The effect of ρ is to rotate the bits of each lane by a length, called the *offset*, which depends on the fixed x and y coordinates of the lane. Equivalently, for each bit in the lane, the z coordinate is modified by adding the offset, modulo the lane size.

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	153	231	3	10	171
$y=1$	55	276	36	300	6
$y=0$	28	91	0	1	190
$y=4$	120	78	210	66	253
$y=3$	21	136	105	45	15

Table 2: Offsets of ρ [8]

The offsets for each lane that result from the computation in Step 3a in Algorithm 2 are listed in Table 2 above.

An illustration of ρ for the case $w=8$ is given in Figure 4 below. The labeling convention for the x and y coordinates in Figure 4 is given explicitly in Figure 2, corresponding to the rows and columns in Table 2. For example, the lane $\mathbf{A}[0,0]$ is depicted in the middle of the middle sheet, and the lane $\mathbf{A}[2,3]$ is depicted at the bottom of the right-most sheet.

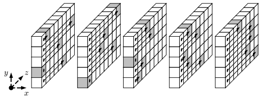


Figure 4: Illustration of ρ for $b=200$ [8]

For each lane in Figure 4, the black dot indicates the bit whose z coordinate is 0, and the shaded cube indicates the position of that bit after the execution of ρ. The other bits of the lane shift by the same offset, and the shift is circular. For example, the offset for the lane $\mathbf{A}[1,0]$ is 1, so the last bit, whose z coordinate is 7 for this example, shifts to the front position, whose z coordinate is 0. Consequently, the offsets may be reduced modulo the lane size; e.g., the lane for $\mathbf{A}[3,2]$, at the top of the left-most sheet, has an offset of 153 mod 8 for this example, i.e., the offset is 1 bit.

3.2.3 Specification of π

Algorithm 3: π(**A**)

Input:
state array **A**.

Output:
state array **A′**.

Steps:

```
1. For all triples $(x, y, z)$ such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let
        $\mathbf{A}'[x, y, z] = \mathbf{A}[(x + 3y) \bmod 5, x, z]$.
2. Return $\mathbf{A}'$.
```

The effect of π is to rearrange the positions of the lanes, as illustrated for any slice in Figure 5 below. The convention for the labeling of the coordinates is depicted in Figure 2 above; for example, the bit with coordinates $x = y = 0$ is depicted at the center of the slice.

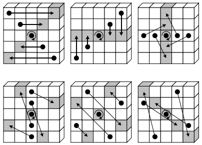
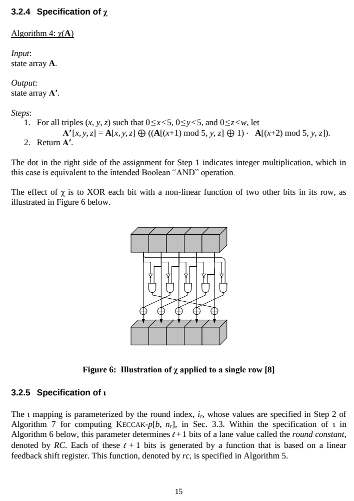


Figure 5: Illustration of π applied to a single slice [8]

Operation



3.2.4 Specification of χ

Algorithm 4: $\chi(\mathbf{A})$

Input:
state array **A**.

Output:
state array **A′**.

Steps:
```
1. For all triples $(x, y, z)$ such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let
       $\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, z] \oplus ((\mathbf{A}[(x+1) \bmod 5, y, z] \oplus 1) \cdot \mathbf{A}[(x+2) \bmod 5, y, z])$.
2. Return $\mathbf{A}'$.
```

The dot in the right side of the assignment for Step 1 indicates integer multiplication, which in this case is equivalent to the intended Boolean "AND" operation.

The effect of χ is to XOR each bit with a non-linear function of two other bits in its row, as illustrated in Figure 6 below.

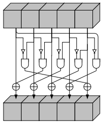
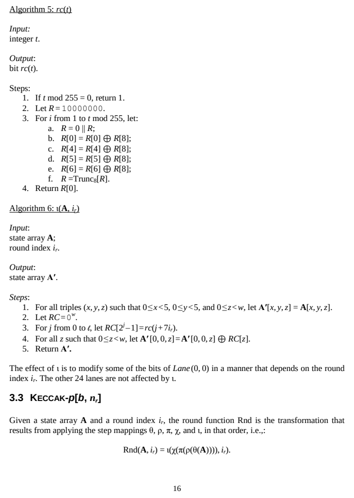


Figure 6: Illustration of χ applied to a single row [8]

3.2.5 Specification of ι

The ι mapping is parameterized by the round index, i_r, whose values are specified in Step 2 of Algorithm 7 for computing KECCAK-$p[b, n_r]$, in Sec. 3.3. Within the specification of ι in Algorithm 6 below, this parameter determines $\ell+1$ bits of a lane value called the *round constant*, denoted by RC. Each of these $\ell + 1$ bits is generated by a function that is based on a linear feedback shift register. This function, denoted by rc, is specified in Algorithm 5.

15



Algorithm 5: $rc(t)$

Input:
integer t.

Output:
bit $rc(t)$.

Steps:
```
1. If $t \bmod 255 = 0$, return 1.
2. Let $R = 10000000$.
3. For $i$ from 1 to $t \bmod 255$, let:
       a. $R = 0 \parallel R$;
       b. $R[0] = R[0] \oplus R[8]$;
       c. $R[4] = R[4] \oplus R[8]$;
       d. $R[5] = R[5] \oplus R[8]$;
       e. $R[6] = R[6] \oplus R[8]$;
       f. $R = \mathrm{Trunc}_8[R]$.
4. Return $R[0]$.
```

Algorithm 6: $\iota(\mathbf{A}, i_r)$

Input:
state array **A**;
round index i_r.

Output:
state array **A′**.

Steps:
```
1. For all triples $(x, y, z)$ such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let $\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, z]$.
2. Let $RC = 0^w$.
3. For $j$ from 0 to $\ell$, let $RC[2^j - 1] = rc(j + 7i_r)$.
4. For all $z$ such that $0 \le z < w$, let $\mathbf{A}'[0, 0, z] = \mathbf{A}'[0, 0, z] \oplus RC[z]$.
5. Return $\mathbf{A}'$.
```

The effect of ι is to modify some of the bits of $Lane(0, 0)$ in a manner that depends on the round index i_r. The other 24 lanes are not affected by ι.

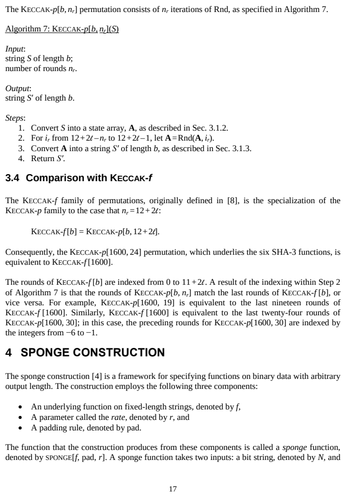
3.3 KECCAK-$p[b, n_r]$

Given a state array **A** and a round index i_r, the round function Rnd is the transformation that results from applying the step mappings θ, ρ, π, χ, and ι, in that order, i.e.,:

$$\mathrm{Rnd}(\mathbf{A}, i_r) = \iota(\chi(\pi(\rho(\theta(\mathbf{A})))), i_r).$$

16

Operation



The KECCAK-$p[b, n_r]$ permutation consists of n_r iterations of Rnd, as specified in Algorithm 7.

Algorithm 7: KECCAK-$p[b, n_r](S)$

Input:
string S of length b;
number of rounds n_r.

Output:
string S' of length b.

Steps:

1. Convert S into a state array, $\mathbf{A}$, as described in Sec. 3.1.2.
2. For i_r from $12+2\ell-n_r$ to $12+2\ell-1$, let $\mathbf{A}=\mathrm{Rnd}(\mathbf{A}, i_r)$.
3. Convert $\mathbf{A}$ into a string S' of length b, as described in Sec. 3.1.3.
4. Return S'.

3.4 Comparison with KECCAK-*f*

The KECCAK-f family of permutations, originally defined in [8], is the specialization of the KECCAK-p family to the case that $n_r = 12+2\ell$:

$$\text{KECCAK-}f[b] = \text{KECCAK-}p[b, 12+2\ell].$$

Consequently, the KECCAK-$p[1600, 24]$ permutation, which underlies the six SHA-3 functions, is equivalent to KECCAK-$f[1600]$.

The rounds of KECCAK-$f[b]$ are indexed from 0 to $11+2\ell$. A result of the indexing within Step 2 of Algorithm 7 is that the rounds of KECCAK-$p[b, n_r]$ match the last rounds of KECCAK-$f[b]$, or vice versa. For example, KECCAK-$p[1600, 19]$ is equivalent to the last nineteen rounds of KECCAK-$f[1600]$. Similarly, KECCAK-$f[1600]$ is equivalent to the last twenty-four rounds of KECCAK-$p[1600, 30]$; in this case, the preceding rounds for KECCAK-$p[1600, 30]$ are indexed by the integers from -6 to -1.

4 SPONGE CONSTRUCTION

The sponge construction [4] is a framework for specifying functions on binary data with arbitrary output length. The construction employs the following three components:

- An underlying function on fixed-length strings, denoted by f,
- A parameter called the *rate*, denoted by r, and
- A padding rule, denoted by pad.

The function that the construction produces from these components is called a *sponge* function, denoted by SPONGE[f, pad, r]. A sponge function takes two inputs: a bit string, denoted by N, and

17