# Education in [Block]Chains

Jonathan A. Poritz

jonathan@poritz.net
www.poritz.net/jonathan
Center for Teaching and Learning and
Department of Mathematics and Physics
Colorado State University-Pueblo

10 June 2019, Domains 2019



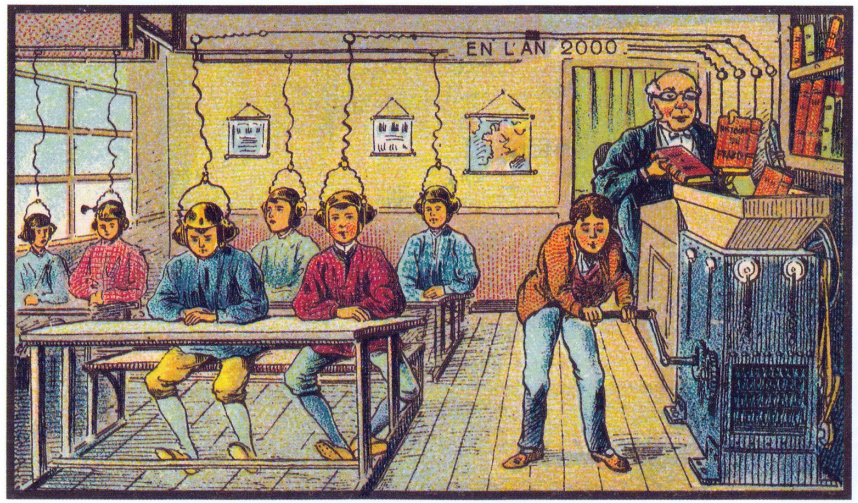
This work is released under a Creative Commons Attribution-ShareAlike 4.0 International License..

*[Note: there is no need to write down URLs or otherwise take specific notes – these slides are shared at*

*poritz.net/j/share/domains19.pdf ... and even that URL will come around again at the end!]*

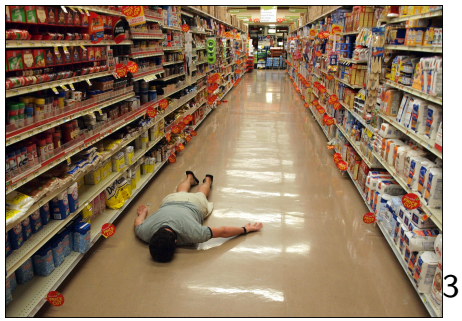
# A Theme of the Indie/Open Edtech World ...

Edtech which does not center on the autonomy and agency of student and instructor is only a tool to turn us all into better – more docile – factory robots and consumers.



At School [1]


[2]


[3]

[1] Jean Marc Cote (if 1901) or Villemard (if 1910) publicdomainreview.org/2012/06/30/france-in-the-year-2000-1899-1910/ [Public domain], via Wikimedia Commons, commons.wikimedia.org/wiki/File:France_in_XXI_Century._School.jpg

[2] Publicity photo of Charlie Chaplin for the film *Modern Times* (1936); [Public domain in the United State (maybe not in other jurisdictions!) since it was published between 1924 and 1977 without a copyright notice], via Wikimedia Commons, commons.wikimedia.org/wiki/File:Chaplin_-_Modern_Times.jpg

[3] *Planking in the supermarket* by TheeErin from www.flickr.com/photos/27073477@N00/4112368718, released under a CC-BY-SA 2.0 license.

# ... Which Drives Much Activity – **Dangerous** Activity?

Hence

- digital literacy education,
- instructors using tools like hypothes.is to shape a web that servers their pedagogy,
- Domain of One's Own projects,
- *etc.*

But the Web can be a horrific place! So....

> *Giving students and instructors autonomy and agency on the World Wild Web without some knowledge of security and privacy is like giving a new driver the car keys and a map of the big city without teaching them how to use their seat belt.*

That's my hobby horse, and I intend to ride it:

 [4]

Today's discussion about **Blockchain** is a case study in this approach.

[4] *Hobby Horse* by Barney Moss via Wikimedia Commons commons.wikimedia.org/wiki/File:Hobby_horse_(6012490195).jpg, released under a CC-BY 2.0 license.

# Origins of the Blockchain Whirlwind: Bitcoin

In October 2009, the pseudonymous *Satoshi Nakamoto* posted this paper to a cryptography mailing list:

He or she shortly thereafter also posted, to a FLOSS sharing site, source code for a reference implementation of the proposed protocol.

**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**

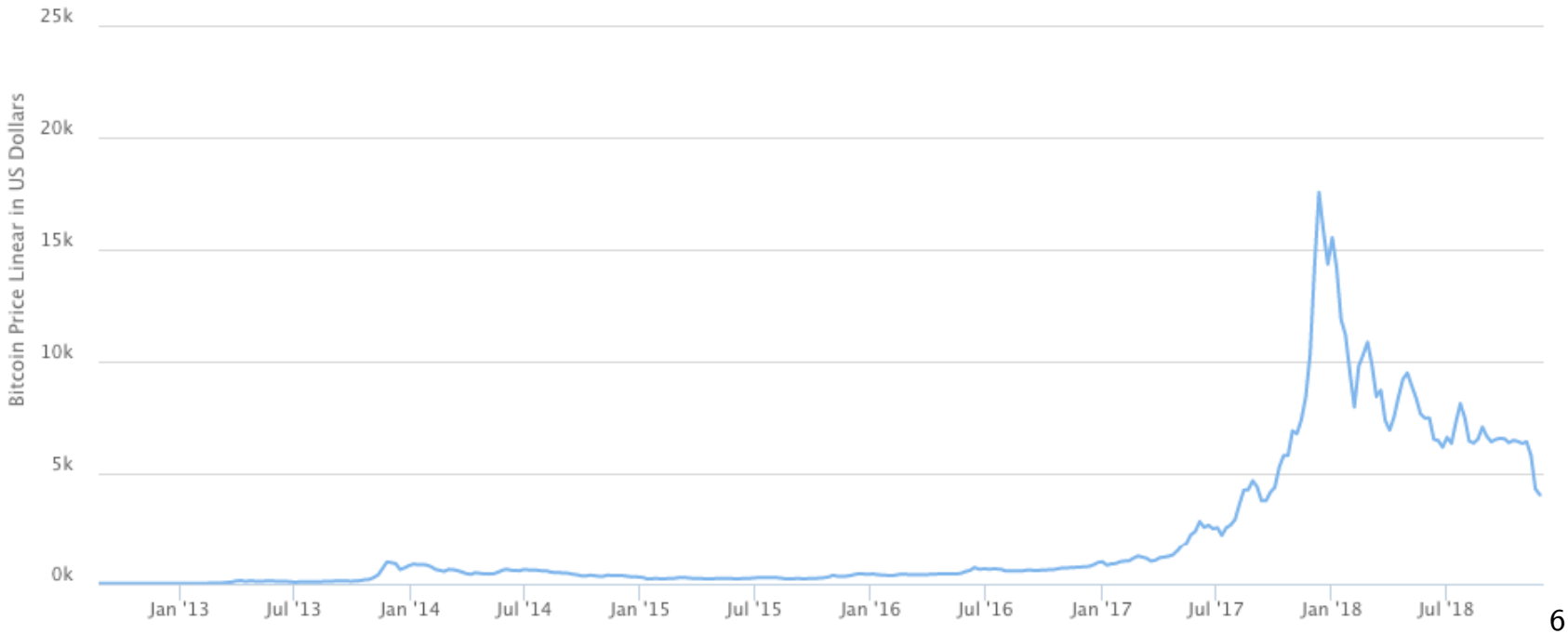
Commerce on the Internet has come to rely almost exclusively on financial institutions serving as [5]

At which point, Yoda could be heard muttering, *"Begun, the Bitcoin War has."*

[5] the full paper is here: bitcoin.org/bitcoin.pdf

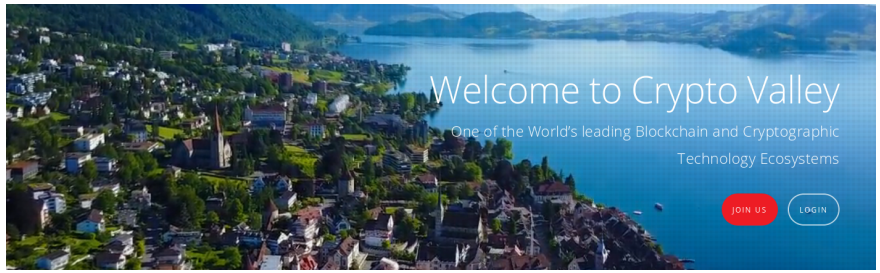
# Some Bitcoin Hype

Price in USD of one ฿




[6]

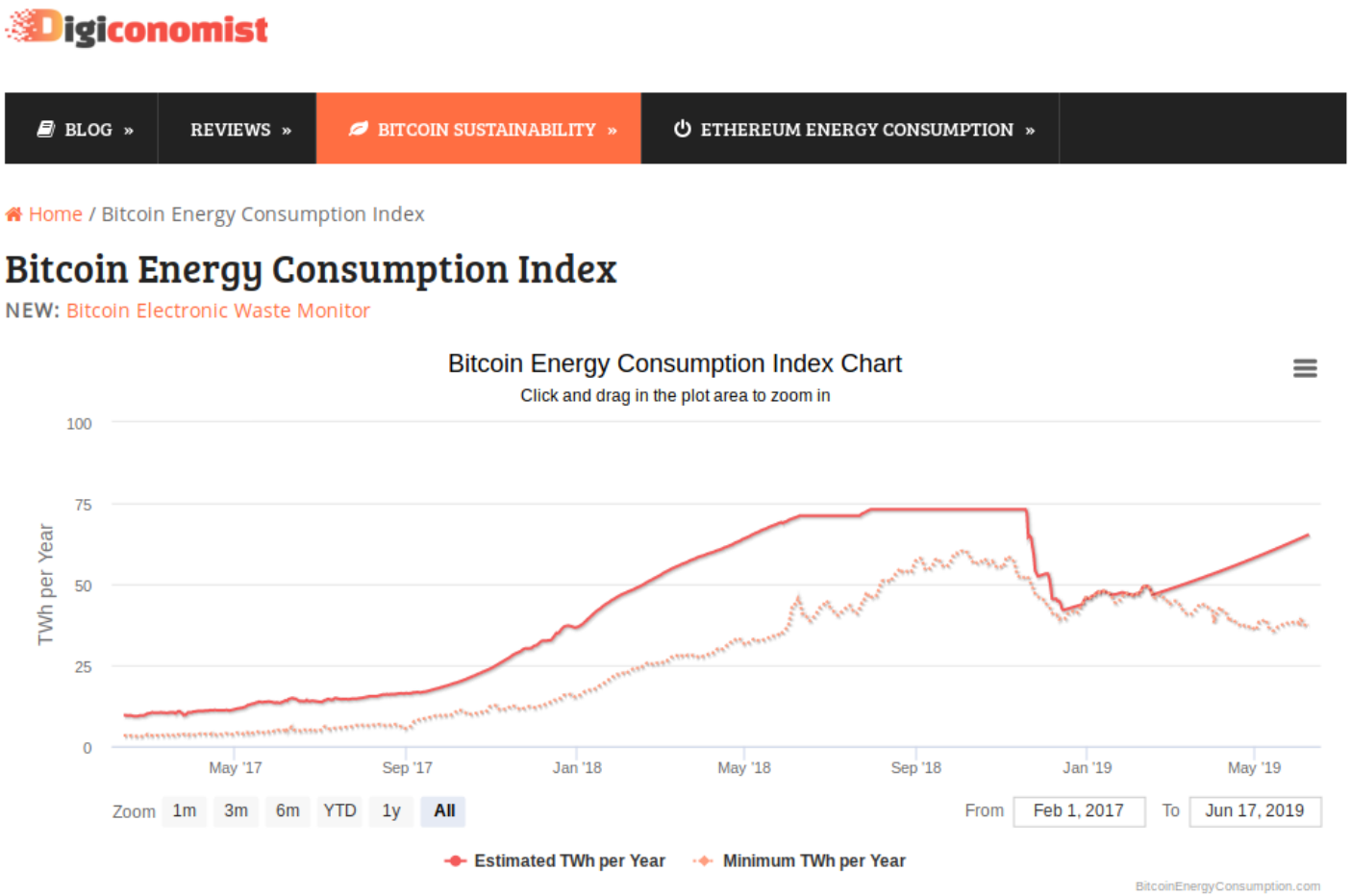
Also nonsense like cryptovalley.swiss:




[6] from bitcoinwiki.org, here.

# A Fly in the Ointment Here, a Few Megatons of $CO_2$, There

Unfortunately:




[7]

For scale, that basically amounts to the total power consumption of **Austria** [the country].

[7] from digiconomist.net, here.

# Diving Into the Deep End: How a Blockchain Works

Why does it consume so much power? I thought you'd never ask.

To answer, we must talk about how bitcoin works. Actually, the underlying technology is called a *blockchain*, so we'll talk for a few minutes about **How a blockchain works.**

Before you stick you fingers in your ears and start humming nervously, let's remember that almost everyone in this room has driven a car on the highway, where a lapse of attention for a fraction of a second could have fatal consequences. And you all have complex, sophisticated disciplinary knowledge. So bear with me for ten fucking minutes.

A blockchain is built out of three basic pieces:

- *digital signatures*,
- a *cryptographic hash function*, and
- a *distributed consensus protocol*.

You're probably already sweating... but chill: every academic discipline and every commercial industry has terms of art. The above three pieces are basically

- *really distinctive personal style* [for digital files],
- a *very effective blender* [that grinds up digital files], and
- an *method for group decision-making* [for groups that meet only on the 'net, never in person].

# References for Cryptology

Crypto tends to intimidate users, because they hear that small mistakes can be fatal. But that is also true of many things in life, and we don't let it stop us.

The other problem with crypto is that it is fairly *mathy*. Which is a feature, not a bug ... to a mathematician, but maybe not to others.[8]

The good news is that there are great free/open resources on the 'net. For example:

> Chapter 4 of my open textbook **Yet Another Introductory Number Theory Textbook** [**YAINTT**], which can be found at poritz.net/jonathan/share/yaintt.pdf [although, being a math textbook, this is rather unabashedly *mathy* – even though that Chapter has a lot of history and terminology.]; or
>
> Ed Felton's [Princeton professor of computer science and Deputy Chief Technology Officer under Obama] **Nuts and Bolts of Encryption: A Primer for Policymakers**, found at https://www.cs.princeton.edu/ felten/encryption_primer.pdf, which is not *mathy* at all.

[8] because the mathematical community has done a terrible job of sharing the joy and beauty of our subject!

# Our protagonists, and an adversary

Most works of cryptology speak of two star-crossed lovers, *Alice* and *Bob*, who attempt to keep the guttering candle of their love alight, though distance separates them and their communications are being monitored by the evil *Eve*.



**Alice**





**Bob**



**Eve**

[Extra credit if you can name the two famous mathematicians who acted as models for these pictures of Alice and Bob.]

# Why is Eve so powerful?

It's important to realize that in many – maybe **most** – situations, it is entirely appropriate to assume that Eve can see all the communication between Alice and Bob while it is in transit. All of the channels you are used to using suffer from this:

> A cell phone is basically a *walkie-talkie with infrastructure* [the infrastructure being all those cell towers all over the place]. Anyone with a radio receiver of the right type who is within the footprint of the same tower can hear the entire exchange. [**Stingrays!**]
>
> Satellite phones are much worse: the footprint is the size of a continent, often.
>
> Anything you do on the Internet is essentially public. [Ask your millennial students how the basic Internet Protocol works – you will be surprised/horrified at what they say.]

Here is a diagram with some basic terminology:

| Alice | on public network | Bob |
|---|---|---|
| **plaintext/cleartext** message $m$ **encrypts** $m$ to $c$ transmits $c$ | $\rightarrowtail$ **ciphertext** $c$ $\rightarrowtail$ | receives $c$ **decrypts** $c$ to recover plaintext $m$ |

# Keys [for symmetric cryptosystems]

If we are to publish our encryption and decryption algorithms, the security must lie in some other secret. This is an additional piece of information called the **key**, which is input into those algorithms, as follows:

| Alice | $\longleftrightarrow$ | private communication of shared key $k \in \mathcal{K}$ | $\longleftrightarrow$ | Bob |
|---|---|---|---|---|
| | | on public network | | |
| message $m_A \in \mathcal{M}$<br>compute $c_A = e_k(m_A)$<br>transmit $c_A$ | | $\rightarrowtail$ ciphertext $c_A$ $\rightarrowtail$ | | receive $c_A$<br>compute $m_A = d_k(c_A)$ |
| receive $c_B$<br>compute $m_B = d_k(c_B)$ | | $\leftarrowtail$ ciphertext $c_B$ $\leftarrowtail$ | | message $m_B \in \mathcal{M}$<br>compute $c_B = e_k(m_B)$<br>transmit $c_B$ |
| *etc....* | | | | |

# Notes on symmetric cryptosystems

The above is called **symmetric** (or **private-** or **secret-key**) **cryptography**. We shall see an alternative in a few minutes.

Notes:

- Both the encryption $e_k$ and decryption $d_k$ use the same key $k$, which must be shared in some private, pre-lapsarian moment. The keyspace $\mathcal{K}$ must be large, otherwise Eve can just try all keys and see which works [which is called a **brute-force attack**].
- Symmetric cryptosystems are fast  you can run a video stream through one without noticing it on a consumer-grade PC.
- The design of symmetric cryptosystems is something of a black art. There is little general theory on the attack or defense side, and the algorithms tend just to be along the lines of *scramble the bits a lot.*
- Some examples:
  - The *Syctale* – ancient Greece
  - The *Caesar cipher* – actually used by Julius Caesar. [addition mod 26...]
  - The *Vigenère Cipher* – thought to be unbreakable for centuries. Easy to break today.
  - The *one-time pad* – completely **unbreakable**; hard to use in practice (but see Leo Marks' **Between Silk and Cyanide: A Code Maker's War 1941-45**)
  - The *Enigma machine* – a German military coding device from WWII.
  - Modern block ciphers like *DES, triple-DES, AES, etc.*

# Asymmetric cryptosystems

If Alice and Bob want to be able to communicate securely without ever having met to exchange the symmetric key, they can instead use **asymmetric** (or **public-key**) **cryptography**:

| Alice | on public network | Bob |
|---|---|---|
| | | pick $k_d \in \mathcal{K}_d$<br>compute $k_e = \mathcal{E}(k_d) \in \mathcal{K}_e$ |
| download $k_e$ | $\longleftarrow$ public key $k_e$ $\longleftarrow$ | publish $k_e$ |
| message $m \in \mathcal{M}$<br>compute $c = e_{k_e}(m)$ | | |
| transmit $c$ | $\longrightarrow$ ciphertext $c$ $\longrightarrow$ | receive $c$ |
| | | compute $m = d_{k_d}(c)$ |

That this is possible at all is very cool. There are a few ways we do it, now, including **RSA** (named after Ron Rivest, Adi Shamir, and Leonard Adelman, who published this idea in 1977) and **elliptic curves** (which are more efficient but less commonly used, since their mathematics is significantly harder to chew that what is behind RSA).

All asymmetric crypto relies upon a mathematical function which is easy to compute in one direction but difficult to invert. For RSA, this is essentially multiplication forward [easy], but factoring backwards [hard]. For other asymmetric algorithms, there are other of these *one-way functions.*

# The "Man-in-the-middle attack"

A significant issue with asymmetric cryptosystems is a **Public-key Infrastructure** [**PKI**], because of the dreaded **man-in-the-middle attack**:

| Alice | Eve | Bob |
|---|---|---|
| | intercept $k_e^B \leftarrowtail$ | generate keys:<br>public $k_e^B$, private $k_d^B$<br>publish $k_e^B$ |
| download $k_e^E$ | generate keys:<br>public $k_e^E$, private $k_d^E$<br>$\leftarrowtail k_e^E$, spoof origin | |
| message $m \in \mathcal{M}$<br>compute $c_A = e_{k_e^E}(m)$<br>transmit $c_A$ | $\rightarrowtail c_A$, intercept | |
| | extract the cleartext<br>$m = d_{k_d^E}(c_A)$<br>change to $m'$ if desired<br>compute $c_E = e_{k_e^B}(m')$<br>spoof origin $c_E \rightarrowtail$ | receive $c_E$ |
| | | read the message<br>$m' = d_{k_d^B}(c_E)$ |

# Digital Signatures

Therefore, we need to be sure that the public keys we use really do belong to the people who we think they do. We do this either by getting the key from someone in person – but that kind of ruins the whole idea of asymmetric crypto! – or we get a key in some way that we are sure of its provenance. One way to be sure would be to have a **digital signature** on the public key, signed by someone whom we trust. Signatures work like this:

| Larry | on public network | Bob |
|---|---|---|
| | | pick $k_d \in \mathcal{K}_d$<br>compute $k_e = \mathcal{E}(k_d)$ |
| download $k_e$ | $\longleftarrow$ public key $k_e$ $\longleftarrow$ | publish $k_e$ |
| | | message $m \in \mathcal{M}$<br>compute $s = d_{k_d}(m)$ |
| receive $(m, s)$ | $\longleftarrow$ signed message $(m, s)$ $\longleftarrow$ | transmit $(m, s)$ |
| if $e_{k_e}(s) = m$<br>ACCEPT<br>otherwise,<br>REJECT | | |

# How to Think Critically About Security/Privacy/Cryptography

Things to think about when learning and critically assessing cryptological gizmos:

- What exact problem was it designed to solve?
  - What [exactly] are the inputs and outputs.
  - What [precise] assumptions are made about *who* knows *what*, *when*?
  - What [precise] assumptions are made about how the adversary will try to break the system – what is the *threat [or attack] model*?
  - What computational power is available to the allies and to the adversaries?
- How confident is the community in the effectiveness of the proposed technique?
  - Is there a mathematical proof, based on just mathematical truths.
  - Is there a mathematical proof that breaking the proposed technique would be equivalent to solving some well-studied problem?
  - Does the security of the proposed technique rest on some engineering factors – *e.g.*, is there a way known to break the technique if we could be a certain new kind of computer?

and, often the most important issue by far:

- How reasonable are the assumptions in a real-world context where we want to use this new technique?

# *E.g.*, Thinking Critically About Digital Signatures

**The problem:** Bob has a secret key with corresponding public key that is reliably known to the whole community as being his. He also has a message he wants to sign.

Bob will transmit to Larry the message along with an extra piece of data, the digital signature. Larry can use the widely known public key associated to Bob, the message, and the signature data to determine if only someone who knew Bob's secret key could have produced that signature on that message.

In the case of RSA-based digital signatures, there is a mathematical proof that breaking this signature scheme would amount to figuring out how to factor large numbers quickly, at which many mathematicians have been unsuccessful for many years.

There is a way to factor large numbers if engineers can build a *quantum computer*. This used to be thought of as far in the future, if ever even possible; now we think it might be 25 years or fewer off. So it will be possible to forge RSA signatures in $\leq$ 25 years.[9]

In the real world, it is very hard to keep a secret key secret forever. It is also very hard to associate public keys to a public identity reliably. Therefore **a robust PKI is a essential** for digital signatures to work in practice, and it is hard to make one and keep it healthy.

[9] Actually, almost all digital signature schemes fall to quantum computers, not just RSA-based ones.

# Cryptographic Hash Functions

A *cryptographic hash function* is an algorithm which takes arbitrarily large chunks of data as input, and produces a *hash* [sometimes called a *digest*] of a certain, fixed size – *e.g.*, the most widely used cryptographic hash function today is called **SHA-256** and it always produces hashes consisting of 256 bits, no matter how big the input.

A hash function is called *pre-image resistant* if it is very hard[10] given any particular possible hash value, to find an input data chunk whose digest has that value.

A hash function is called *second pre-image resistant* if it is very hard, given one input data chunk, to find a second which hashes to the same digest as that chosen first chunk.

Basically, a hash function is a blender into which you pour data, which blends all its input thoroughly and produces a small output pellet. The pellet is almost unique to that set of ingredients, in the sense that it would take many lifetimes of the universe for you to find another set of input ingredients which would yield the same output pellet.

**The problem** is described above. Known hashing algorithms are quite complex and have little mathematical structure. Therefore, it is very hard to attack them... but, also, there are not any proofs that they do what we hope they will. Old hash functions, for example, are no longer used because we now know how to break them.

[10] meaning that, essentially, one should just have to keep trying random inputs until one gets lucky.

# Hash Functions As Message Digests and For Chains

Suppose I send you a big file and we want to be sure it didn't get corrupted in transit. I could compute a hash of that data on my end, you could do the same on yours, and I could read you the digest over the phone – if they agree, we'd be quite certain the files agree as well.

You can use hash functions to make **immutable chains**. Suppose we all agree on some block of starting data, called the *genesis block* in the blockchain community. Then we start attaching new blocks to the list of official, on-chain blocks, by some sort of distributed agreement process, **and also** putting the hash of the previous block into each new block on the chain – this is called a *hash chain*. If, afterwards, we have a dispute about what the whole chain is, we could each run down the whole chain, checking that each hash value in the $n^{th}$ block is equal to the hash of the $(n-1)^{st}$ block. Since it is hard to find second-preimages for good hash functions, this is impossible to forge, and thus it is impossible to change the old blocks on an existing chain: hash chains are immutable.

# Hash Functions to Slow Computers Down

Suppose you want to slow someone down. You could had them a chunk of data and ask them to find some other piece of data to add to that such that when they hashed the big, joined data chunk, the output hash would end in at least five (or seven or ten or whatever, depending upon how much you want to slow them down) 0s.

You couldn't ask for them to find data which will make the hash value exactly equal some given value, but instead this task of working to get a certain number of 0s can be done, only with a lot of computation, by just trying over and over again.

The bitcoin protocol uses this method to slow down – and randomly assign a winner, whoever found the second piece of data which makes the whole thing have a hash ending in a certain number of 0s – all of the people working around the globe to verify transactions made on the bitcoin network. The number of 0s needed to win this competition can be changed as the total amount of computational power on planet Earth dedicated to doing this (stupid, useless, but un-short-cut-able) work increases, so that even with all that power, there will only be winner about once every ten minutes.

The total hashing power on planet Earth right now can do about 55 million trillion SHA-256 hashes per second – that's $5.5 \times 10^{19}$ hashes per second. Much if this is in special hardware in hashing farms in China....

# Distributed Consensus

Suppose you have a block of transactions – maybe people submitting records of buying or selling goods by transferring some artificial *cryptocurrency* among users – you want to submit to a global, public ledger. It is important that everyone agrees upon what are the valid blocks, because otherwise a bad actor could spend some cryptocurrency on one transaction in one block, and the same cryptocurrency in another transaction with someone else: this is **the double-spend problem**.

The easiest way for this to work is if there is a central party who collects all proposed blocks and simply decides which ones will go on the chain, perhaps by first-come-first-served, perhaps by playing favorites (or so people fear). Such a central decision-maker is called a *trusted third party [TTP]* by cryptologists. Distributed consensus is very easy, fast, and efficient with a TTP.

Without a TTP, as bitcoin tries to go, much more work is required. The bitcoin approach is to make everyone do the "hash to get a certain number of 0s" game, and then publish their winning extra bit of junk data. Whoever wins can prove it, with that data, and gets to submit their preferred block to the global public ledger.

Sometimes, two people will win at the same time in different places, and the chain will split. But if half of the network works on growing one chain and half works on the other, pretty quickly it will be very likely that one will be quite a bit longer than the other, and everyone will go over to the longer valid chain at that point.

# Wrapping Up The Blockchain Bow

The term **blockchain** is used to describe a hash chain whose globally recognized individual blocks are agreed upon by a distributed consensus protocol.

Generally, those who can submit blocks come from a very open group – potentially *anyone*[11] – and follow some internal rules such as for maintaining a cryptocurrency ledger or running an on-chain Turing-complete programming language [*Ethereum*, a company based in the Crypto Valley, calls these on-chain programs **smart contracts** in an essentially meaningless bit of excellent marketing].

Another differentiator between different blockchains is how the individuals in the group of participants identify themselves. Generally, the choice is with some identity strictly coupled (hopefully) to a real-world individual or entity **or** one where individuals act through pseudonymous (hopefully[12] IDs. Both require some kind of reliable PKI, however!

Different blockchains use different consensus protocols: bitcoin's *proof of work* has a horrible carbon cost, is slow, *etc.* Another protocol is based on *proof of stake*, which is less energy-intensive but amounts to formalizing a strategy of **the rich always get richer**.

[11] Some people restrict this group and then talk about *permissioned blockchains*, but I think that is pretty much the same as *freeze-dried water*.

[12] This has turned out to be much harder that it seemed it would be, *e.g.*, in bitcoin's case.

# Thinking Critically About Blockchains

Whatever particular version of a blockchain, the solution it provides [in the spirit of our *Thinking Critically About Security...* strategy above] involves these key features:

- A chain of linked, sequential **public** records.
- This [hash] chain is **immutable**.
- Individuals participate in the blockchain through either pseudonymous or intentionally real world-linked IDs supported by a **robust PKI**.
- The progress of the blockchain is realized by a distributed consensus protocol which avoids putting its trust in any particular [third] party.

In short, blockchains are characterized as

- **public**
- **immutable**
- relying on a **robust PKI**
- eschew **TTP**s.

The constituent cryptographic pieces that realize these characteristics may or may not be generally thought of as reliable, but the ways these blocks are combined is not much subject to debate.

Whether the above characteristics are reasonable assumptions in any particular real-world context is much more questionable....

# Blockchains ... Of Limited Use?

The integrity of the bitcoin ledger is based on a huge amount of (useless) work, needed because cryptocurrency fanatics don't trust the government: it's all about the TTP.

Now what about other use-cases?

**Educational institutions should "put student credentials on the blockchain:" it's public, immutable, and not subject to any single party's control.**
*But this is nonsense.* There is a natural TTP for a credential: the issuing institution! OK, we need a functioning PKI, but so does a blockchain-based credential system (the credential would have to be signed, in its form "on the chain," as well).

**Walmart will "put its supply chain on the blockchain."**
*But this is nonsense.* There is a natural TTP for a Walmart's supply chain: Walmart trusts Walmart!

**Marijuana producers in Colorado will put their production records on a blockchain** so that the state marijuana oversight organization will be able to check that it has all been produced according to state law.
*But this is nonsense.* There is a natural TTP here as well: that state supervisory agency! Also, just having a cryptographically verified blockchain doesn't mean that the data entered in its blocks corresponds correctly with the real world: this is a misreading of the idea of "trust" here.

# The Moral(s)

**Use your critical thinking skills** even with the very mathy cryptology, privacy, and security.

One outline of where to get started in that process is the basics of

- What exact problem was it designed to solve?
- How confident is the community in the effectiveness of the proposed technique?
- How reasonable are the assumptions in a real-world context where we want to use this new technique?

and particularly the sub-parts of those questions as mentioned before.

*You can think through those questions with a very low level of* **mathiness**, *and be highly effective in understanding proposed cryptology/security/privacy approaches* **in context**.

**Get some comfort level with basic crypto/security/privacy**, it's your best seat belt on the Information Superhighway. **Share that knowledge and comfort** (or at least willingness to struggle for comfort) **with your students,** otherwise what kind of horrible mentor are you to people about to drive those dangerous ways?

# Questions, Comments, and Contact Info

## Questions? Comments?

Also feel free to contact me at jonathan@poritz.net .

Get these slides at poritz.net/j/share/domains19.pdf and all files for remixing[13] at poritz.net/j/share/domains19/ .

If you don't want to write down that full URL, just remember
poritz.net/jonathan/share
or poritz.net/j/share
or poritz.net/jonathan [then click **Always SHARE**]
or poritz.net/j [then click **Always SHARE**]
or scan ⟶ [then click **Always SHARE**]



[13] subject to CC-BY-SA