

Security: An Overview of Cryptographic Techniques

14-736

With slides from: Debabrata Dash, Nick Feamster,
Gregory Kesden, Vyas Sekar and others

Cryptography, Cryptographic Protocols and Key Distribution

- Authentication
- Mutual Authentication
- Private/Symmetric Keys
- Public Keys
- Key Distribution

What do we need for a secure communication channel?

- Authentication (Who am I talking to?)
 - Confidentiality (Is my data hidden?)
 - Integrity (Has my data been modified?)
-
- Availability (Can I reach the destination?)

What is cryptography?

"cryptography is about communication in the presence of adversaries."

- Ron Rivest

"cryptography is using math and other crazy tricks to approximate magic"

- Unknown TA

What is cryptography?

Tools to help us build secure communication channels that provide:

- 1) Authentication
- 2) Integrity
- 3) Confidentiality

Cryptography As a Tool

- Using cryptography securely is not simple
- Designing cryptographic schemes correctly is near impossible.

Today we want to give you an idea of what can be done with cryptography.

Take a security course if you think you may use it in the future

The Great Divide

Symmetric Crypto
(Private key)
(E.g., AES)

Asymmetric Crypto
(Public key)
(E.g., RSA)

Shared secret
between parties?

Yes

No

Speed of crypto
operations

Fast

Slow

Symmetric Key: Confidentiality

Motivating Example:

You and a friend share a key K of L random bits, and want to secretly share message M also L bits long.

Scheme:

You send her the $xor(M, K)$ and then she “decrypts” using $xor(M, K)$ again.

- 1) Do you get the right message to your friend?
- 2) Can an adversary recover the message M ?
- 3) Can adversary recover the key K ?

Symmetric Key: Confidentiality

- One-time Pad (OTP) is secure but usually impractical
 - ♦ Key is as long as the message
 - ♦ Keys cannot be reused (why?)

In practice, two types of ciphers are used that require constant length keys:

Stream Ciphers:

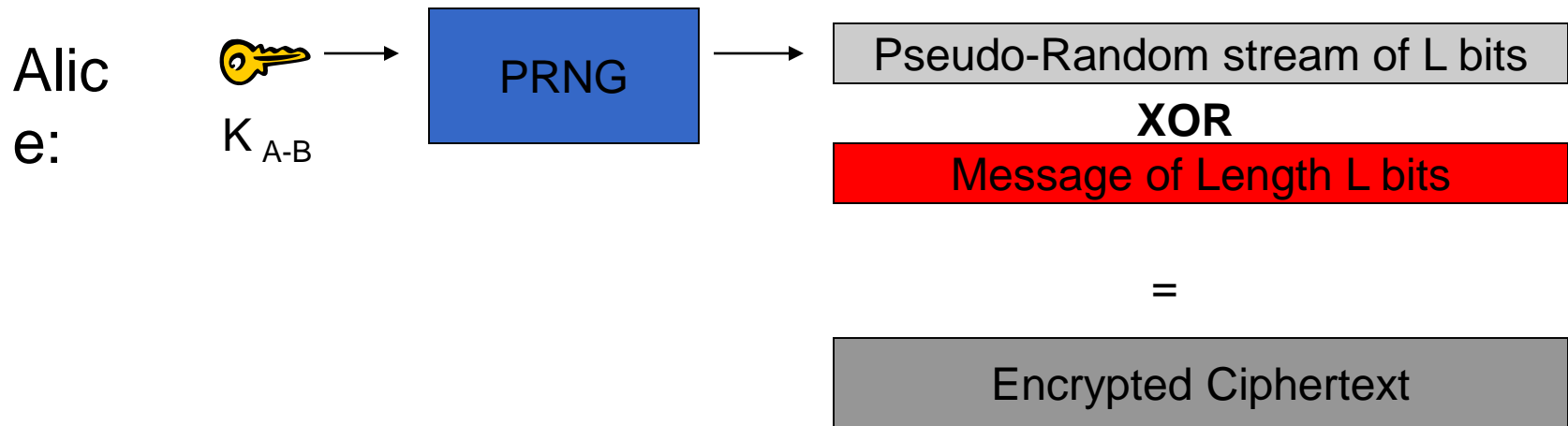
Ex: RC4, A5

Block Ciphers:

Ex: DES, AES,
Blowfish

Symmetric Key: Confidentiality

- Stream Ciphers (ex: RC4)



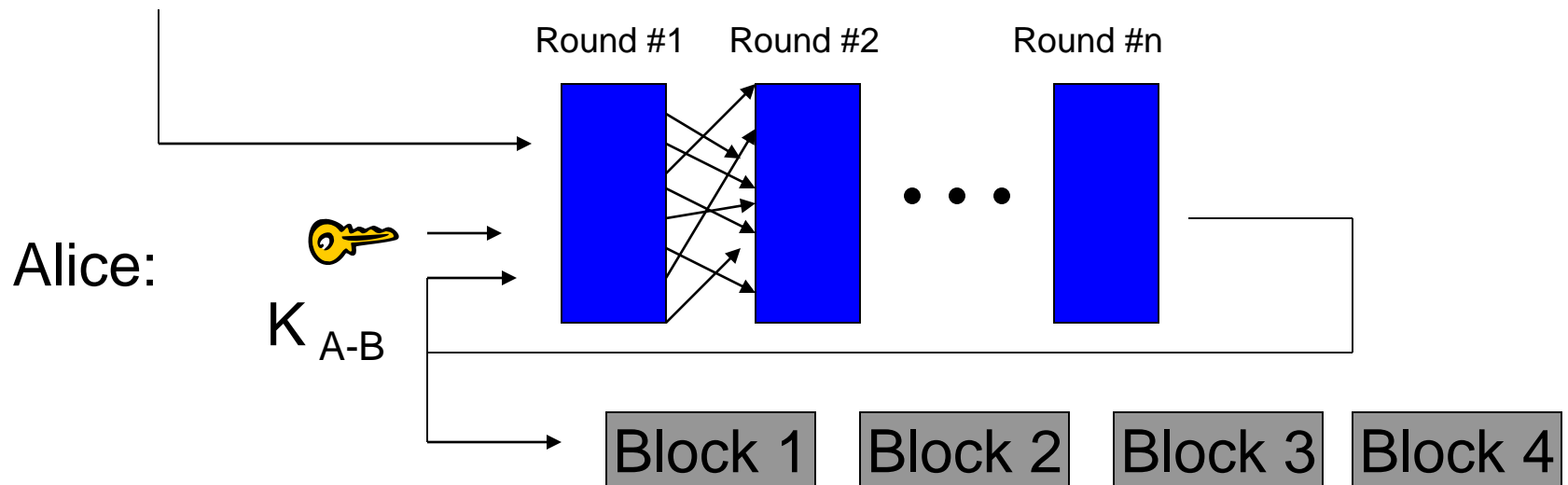
Bob uses K_{A-B} as PRNG seed, and XORs encrypted text to get the message back (just like OTP).

Symmetric Key: Confidentiality

■ Block Ciphers (ex: AES)

Block 1 Block 2 Block 3 Block 4

(fixed block size,
e.g. 128 bits)



Bob breaks the ciphertext into blocks, feeds it through decryption engine using K_{A-B} to recover the message.

Cryptographic Hash Functions

- Consistent

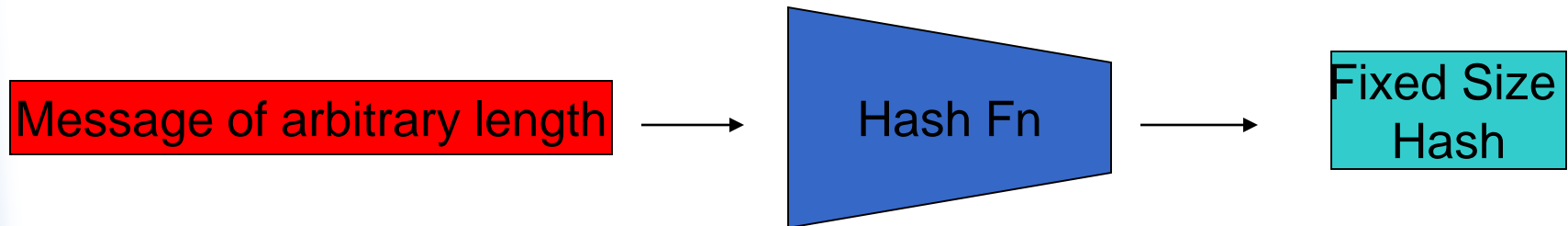
hash(X) always yields same result

- One-way

given Y, can't find X s.t. $\text{hash}(X) = Y$

- Collision resistant

given $\text{hash}(W) = Z$, can't find X such that $\text{hash}(X) = Z$

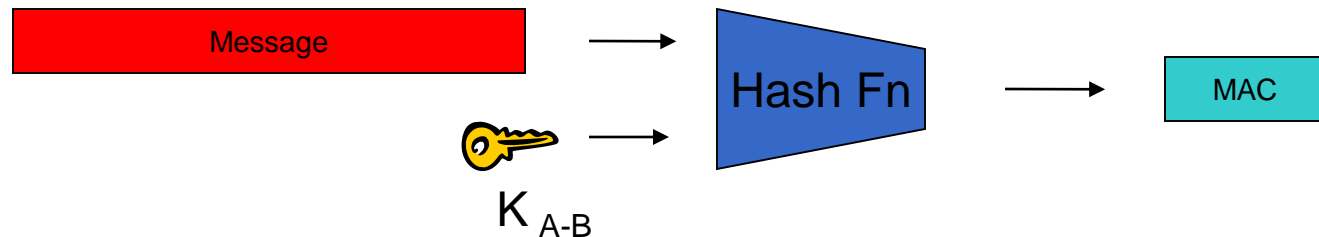


Symmetric Key: Integrity

- Hash Message Authentication Code (HMAC)

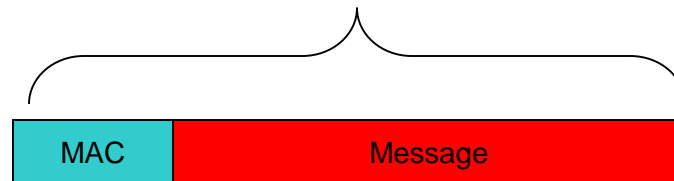
Step #1:

Alice creates
MAC



Step #2

Alice Transmits Message & MAC



Step #3

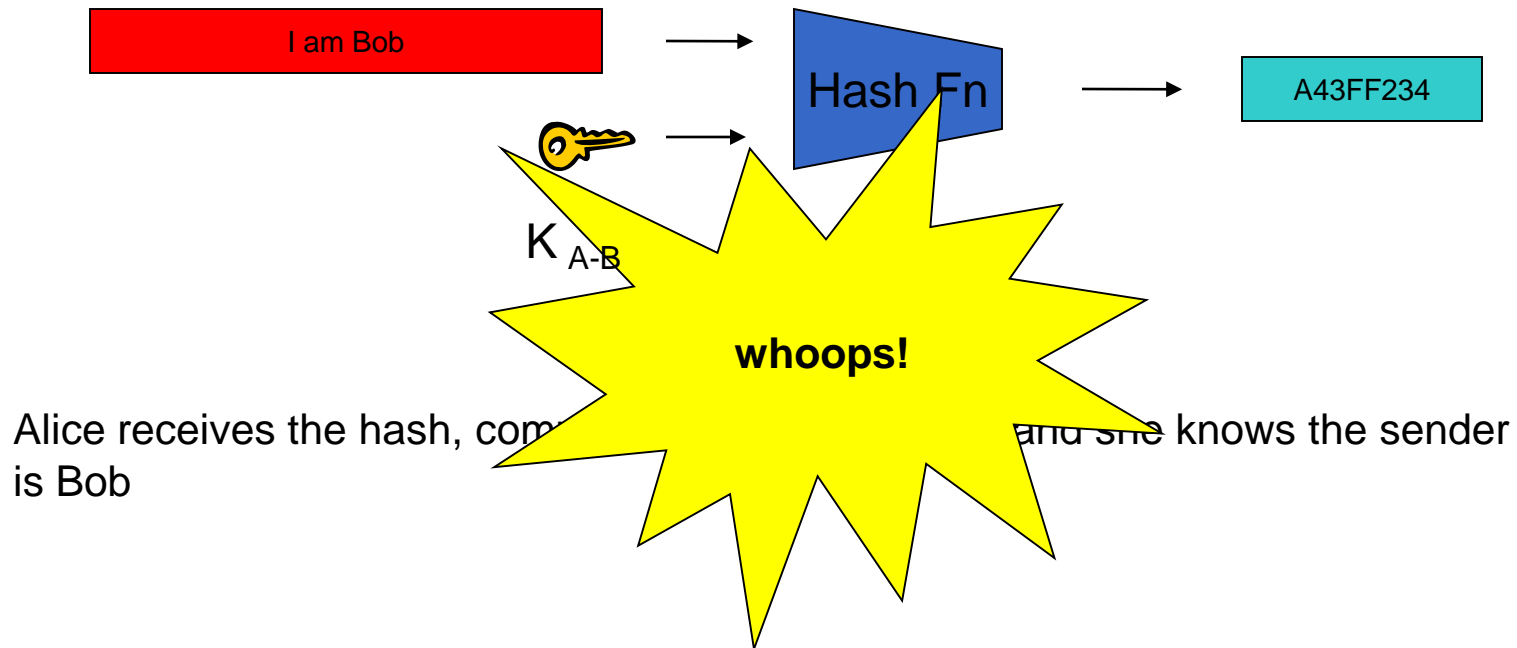
Bob computes MAC with
message and K_{A-B} to verify.

Why is this secure?

How do properties of a hash function help us?

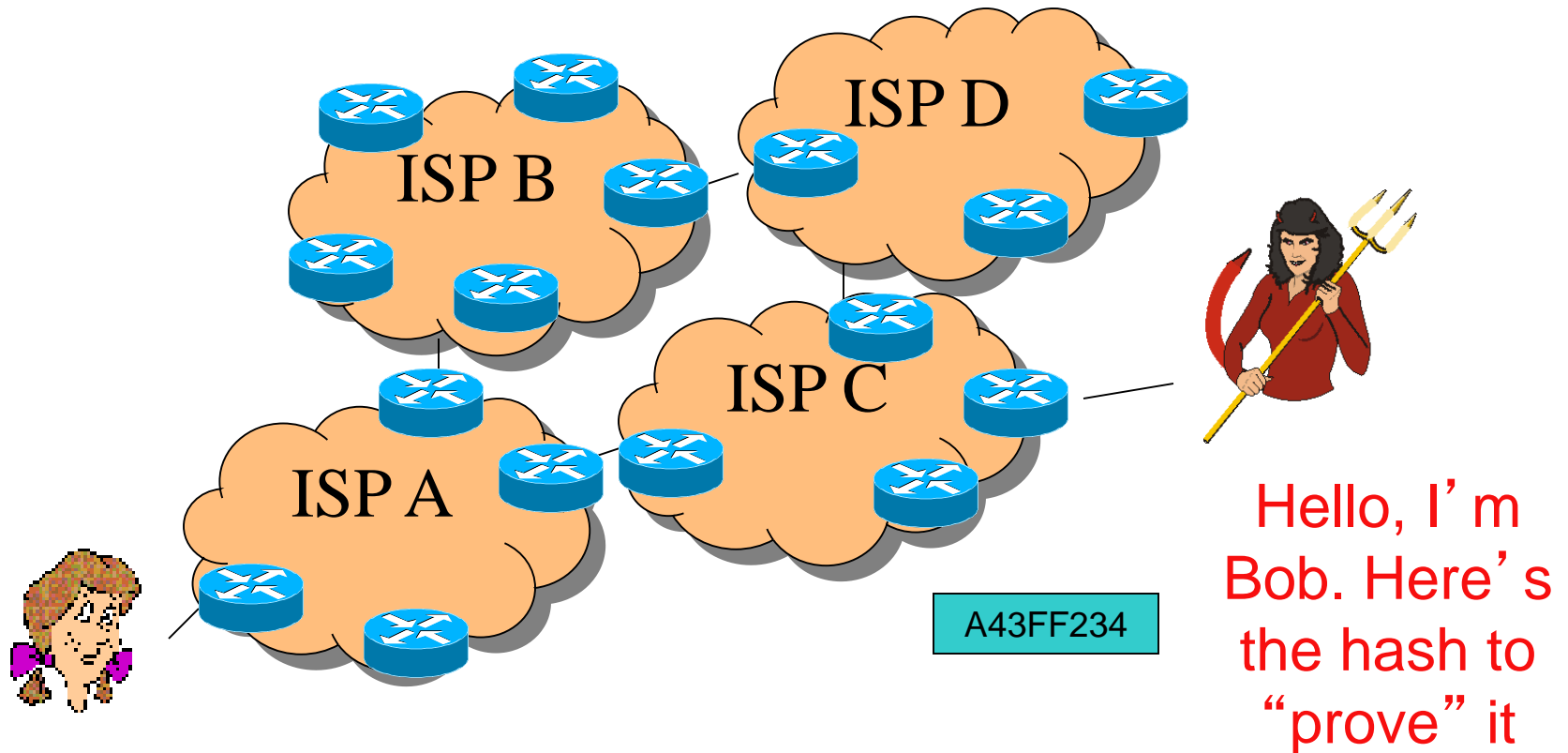
Symmetric Key: Authentication

- You already know how to do this!
(hint: think about how we showed integrity)



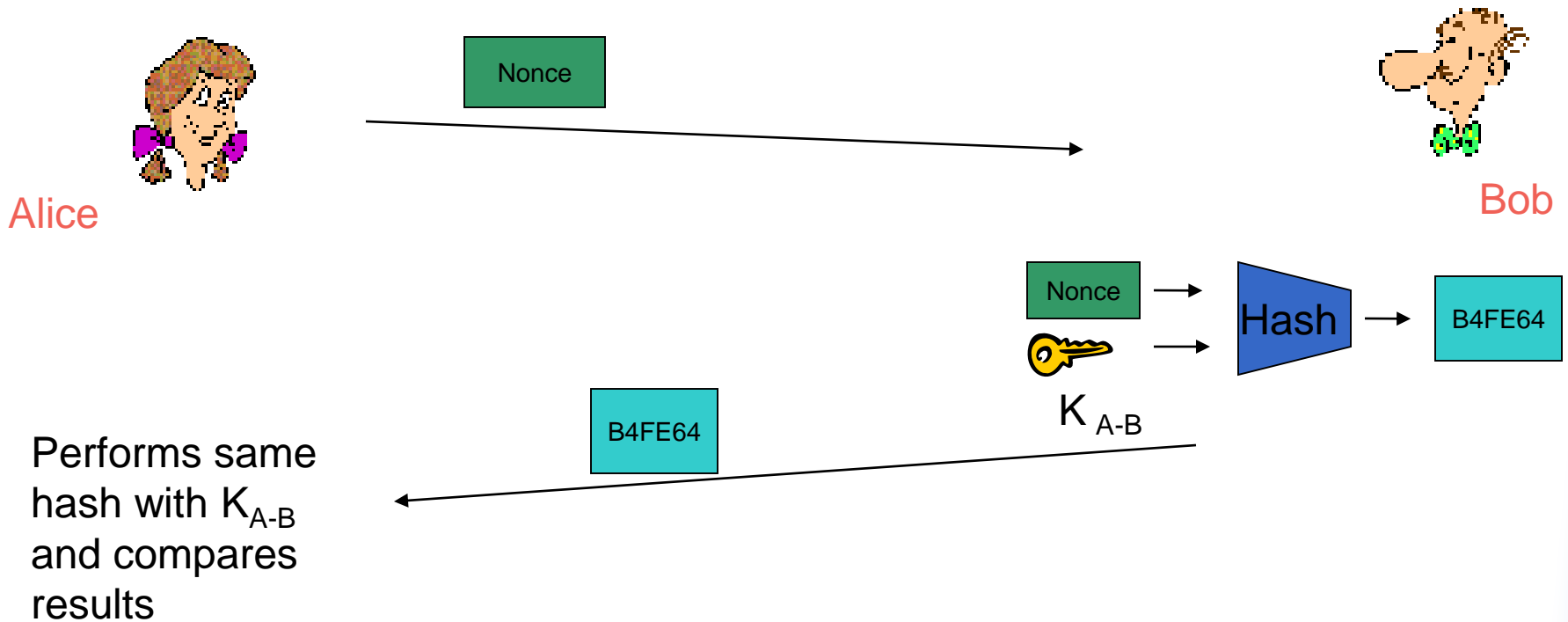
Symmetric Key: Authentication

What if Mallory overhears the hash sent by Bob, and then “replays” it later?



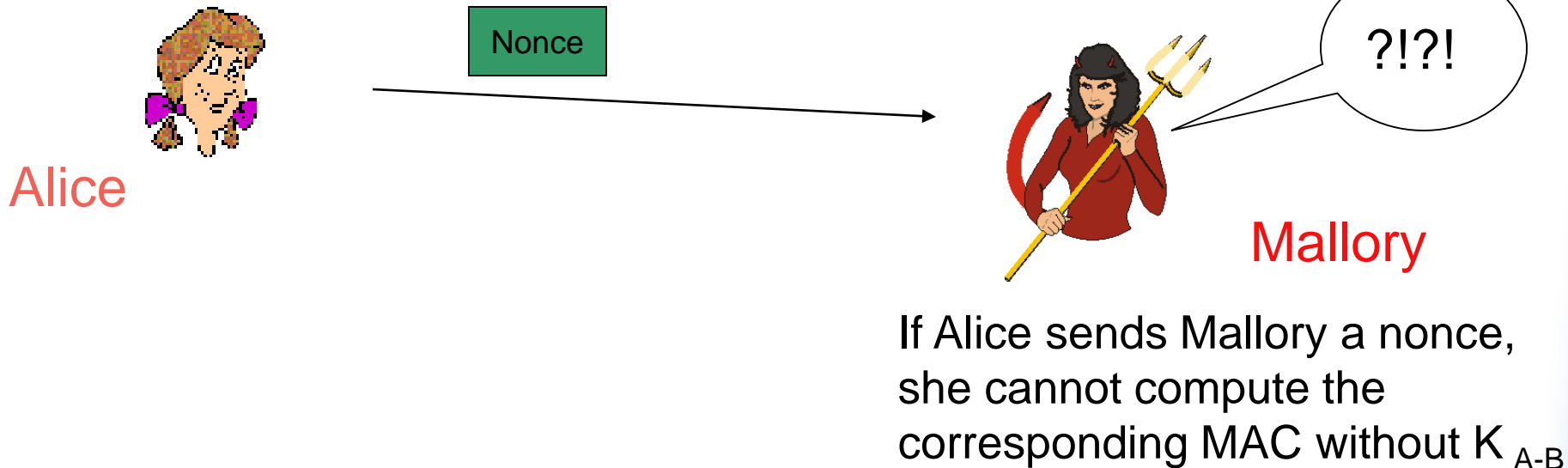
Symmetric Key: Authentication

- A “Nonce”
 - ♦ A random bitstring used only once. Alice sends nonce to Bob as a “challenge”. Bob Replies with “fresh” MAC result.



Symmetric Key: Authentication

- A “Nonce”
 - ♦ A random bitstring used only once. Alice sends nonce to Bob as a “challenge”. Bob Replies with “fresh” MAC result.



Symmetric Key Crypto Review

- Confidentiality: Stream & Block Ciphers
- Integrity: HMAC
- Authentication: HMAC and Nonce

Questions??

Are we done? Not Really:

1) Number of keys scales as $O(n^2)$

2) How to securely share keys in the first place?

Asymmetric Key Crypto:

- Instead of shared keys, each person has a “key pair”



K_B

Bob's public key



K_B^{-1}

Bob's private
key

- The keys are inverses, so: $K_B^{-1} (K_B (m)) = m$

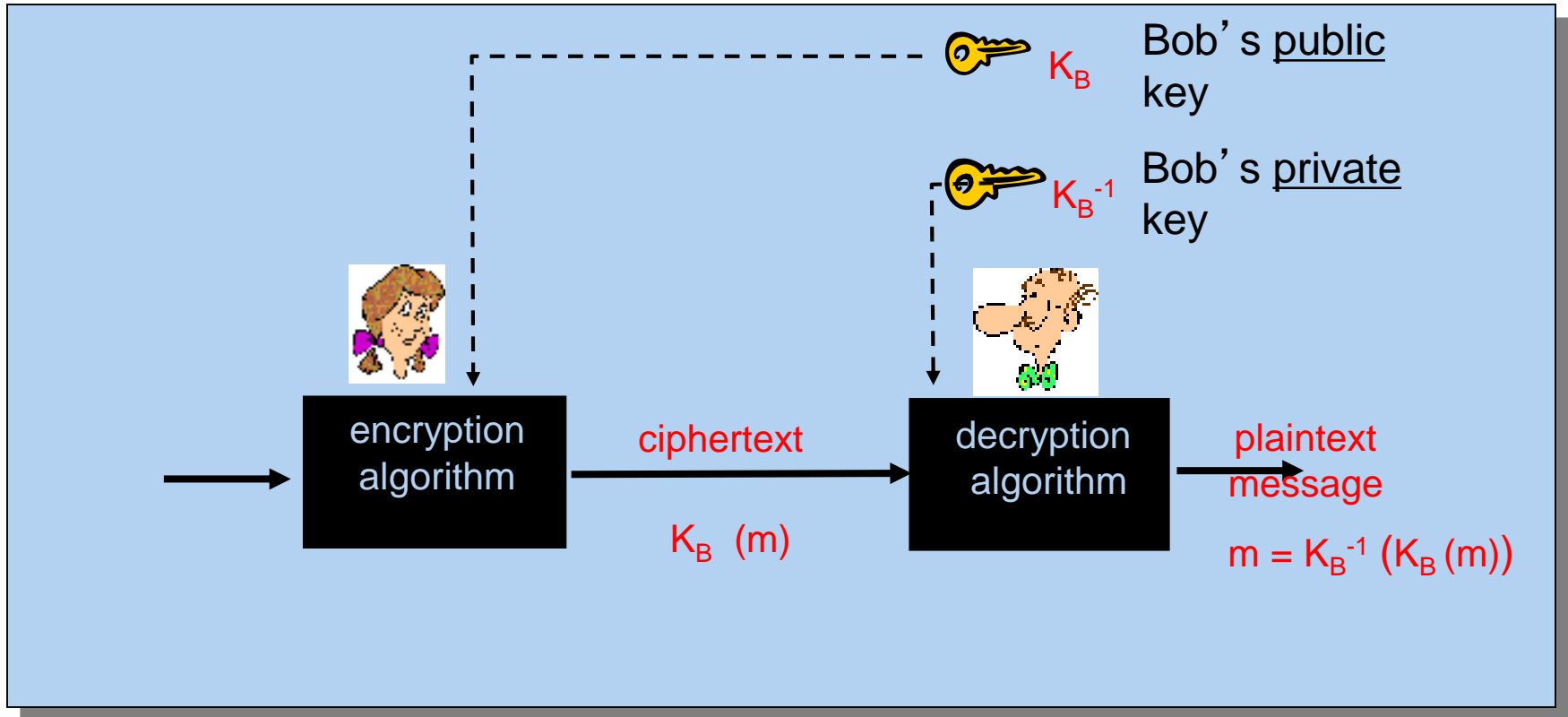
Asymmetric Key Crypto:

- It is believed to be computationally unfeasible to derive K_B^{-1} from K_B or to find any way to get M from $K_B(M)$ other than using K_B^{-1} .

=> K_B can safely be made public.

Note: We will not explain the computation that $K_B(m)$ entails, but rather treat these functions as black boxes with the desired properties.

Asymmetric Key: Confidentiality



Asymmetric Key: Sign & Verify

- If we are given a message M , and a value S such that $K_B(S) = M$, what can we conclude?
- The message must be from Bob, because it must be the case that $S = K_B^{-1}(M)$, and only Bob has K_B^{-1} !
- This gives us two primitives:
 - $\text{Sign}(M) = K_B^{-1}(M) = \text{Signature } S$
 - $\text{Verify}(S, M) = \text{test}(K_B(S) == M)$

Asymmetric Key Review:

- Confidentiality: Encrypt with Public Key of Receiver
- Integrity: Sign message with private key of the sender
- Authentication: Entity being authenticated signs a nonce with private key, signature is then verified with the public key

But, these operations are computationally expensive*

Biometrics

- Nice in some respects
 - No need to distribute
 - Reducible to digital form
 - Unique in practice
- Hard to duplicate?
 - Used via binary representation
 - Warm gelatin fingers or slip-on finger-pads molded to prints?
 - Artificial eyeballs made to match scans?
 - Pictures? Videos w/blinking?
- Change over time?
 - Injury?
 - Aging?
- **Not replaceable or revocable**
 - What happens when “stolen?”
 - Are you “Deleted”?!?!?
 - (Well, you do have 10 fingers, two retinas, one nose, etc)

Multi-Factor, Human Factors

- Best systems use more than one factor
 - Something you know
 - Something piece of you
 - Biometrics + Password/Q&A Challenge, Etc
 - More natural factors better than fewer unnatural challenges
 - More weak factors may be stronger than fewer stronger factors
- Human factors are critical
 - Too many password restrictions? Too many passwords?
 - Write them down on Post-Its Notes!

Summary

- Symmetric (pre-shared key, fast) and asymmetric (key pairs, slow) primitives provide:
 - Confidentiality
 - Integrity
 - Authentication
- “Hybrid Encryption” leverages strengths of both.
- Great complexity exists in securely acquiring keys.
- Crypto is hard to get right, so use tools from others, don't design your own (e.g. TLS).

Introduction to Blockchains

John Kelsey, NIST

https://csrc.nist.gov/csrc/media/projects/supply-chain-risk-management/documents/ssca/2016-fall/wed_am2-block_chain_john_kelsey.pdf

Overview

- Prologue: A chess-by-mail analogy
- What problem does a blockchain solve?
- How do they work?
 - Hash chains
 - Deciding what blocks are valid on the chain
 - Deciding whether we have the current chain
- Permissioned blockchains, proof of work, etc.
- Wrapup

Warm-up: Alice and Bob want to play chess by mail

- Alice sends Bob “1 e4”
- Bob sends back “1 ... e5”
- Alice sends Bob “2 Nf3”
- ...
- Each of these messages is one move in the game
- What’s necessary for them to be able to play the game?

They have to agree on the state of the board

If they don't agree on the state of the board, they can't play a game!

1. Both know the starting positions of the board.
2. Both know the sequence of messages so far.
 - Those messages make up a transcript of the game.
3. Thus, they can reconstruct the state of the board.

If we agree on history, we agree on the present state of the world!

What's that got to do with blockchain?

- We have some distributed system
- We need to all agree on the state of some system
- We all agree on the initial state of the system
- A blockchain contains a history of individual transactions
- Thus: We can all agree on the current state of the system

**A blockchain lets mutually-distrusting entities agree on history...
...which lets them agree on the state of the system now.**

Why is this important?

- Example: Bitcoin
- Suppose I want to transfer 100 BTC to you.
- You need to know whether my account has 100 BTC in it.
- For that, you need to know the current state of the system.
- Note: You need to know the **current** state
 - If you're looking at an old state of the system, I might be paying you with money I've already spent!

What problem does a blockchain solve?

A blockchain lets us agree on the state of the system, even if we don't all trust each other!

- **Ultimate goal: We all need to agree on the state of some system.**
 - How much BTC in each account?
 - Who owns which property?
 - What's the current state of my program?
- **We can all agree on that if we agree on history.**
 - Starting state + history → current state
- **We don't want a single trusted arbiter of the state of the world.**
 - We want some level of decentralization—not a single point of failure or compromise.

Trusted Arbiter

- If we had a completely trusted arbiter, we wouldn't need a blockchain!
- We could just define reality as whatever TA said it was.
- For a payment system, imagine TA as the bank
 - Bank provides the official sequence of transactions and account balances
 - When you want to spend your money, you send a message to bank
 - Bank permits transaction if you have money, and updates account balances.

Why not just have a trusted arbiter, then?

1. Single point of failure
 - If the TA goes down for a week, the system stops working!
2. Concentration of power
 - “He who controls the past, controls the future”
 - TA can censor transactions, impose new conditions to get transactions included in history, etc.
3. Maybe there's nobody we all trust

So what does a blockchain buy us, again?

- Distributed system
- We don't all trust each other or any single entity
- We want to agree on history
- ...so we can agree on the state of our system...
- ...so we can do something.

We get the functionality of a trusted arbiter...

...without needing a trusted arbiter

How does it work?

- **A blockchain is a sequence of hash-chained records**
 - Once you've seen record N, you can't change anything in the past.
- **Some procedure for adding blocks to blockchain**
 - Who gets to add blocks? How is it done?
- **Validity conditions for new blocks**
 - Are transactions valid? Are digital signatures correct? Etc.
 - Enforced by consensus—chains with invalid blocks won't be accepted.
- **Some procedure for deciding between alternative candidate blockchains.**
 - When Alice and Bob have different pictures of history, there's some way for them to eventually come to agreement about who is right.

Building Block: Cryptographic hash functions

A ***cryptographic hash function***:

- Takes any bitstring as an input* (Like a 10 MB file)
- Produces a fixed-length output (Typically 256 or 512 bits)
- Nobody can find ***collisions***.
- ***Examples***: SHA256, SHA512, SHA3-256, RIPEMD-160

* Sometimes there's a (really huge) maximum input length.

What's a collision?

- Suppose I can find two different inputs X and Y so that
$$\text{Hash}(X) = \text{Hash}(Y)$$
- That's a collision.
- For a cryptographic hash function to be any good, it needs to be ***collision-resistant***.
- That just means it's impossible in practice to find colliding inputs.

Why is collision resistance useful?

- If nobody can find $X \neq Y$ such that $\text{Hash}(X) == \text{Hash}(Y)$,
- ...then we can use $\text{hash}(X)$ as a kind of **message digest** of X .
 - Digital signatures actually sign $\text{hash}(\text{message})$ instead of message.
- Nobody can change X without changing $\text{hash}(X)$
 - If they could do that, they can find collisions for $\text{hash}()$
- $\text{hash}(X)$ also **commits** to X .
 - Once I've seen $\text{hash}(X)$, later, you can show me X , and I'll know it's the value you committed to
 - ...you can't show me some other X^* , because it won't have the same hash.

Building block: Hash chains

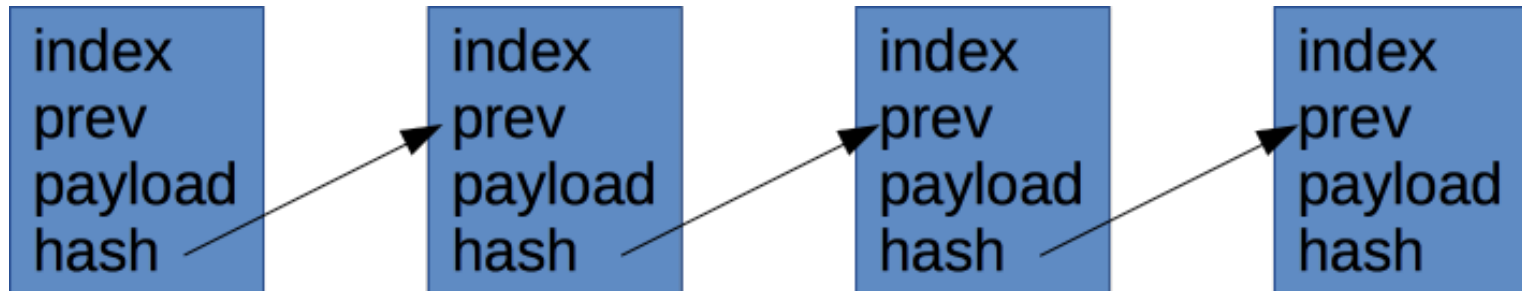


Figure: A sequence of records linked together; each record contains the hash of the previous record.

What's a hash chain?

- A *hash chain* is a sequence of records in which each record contains the hash of the previous record in the chain, and the hash of all the current record's contents.

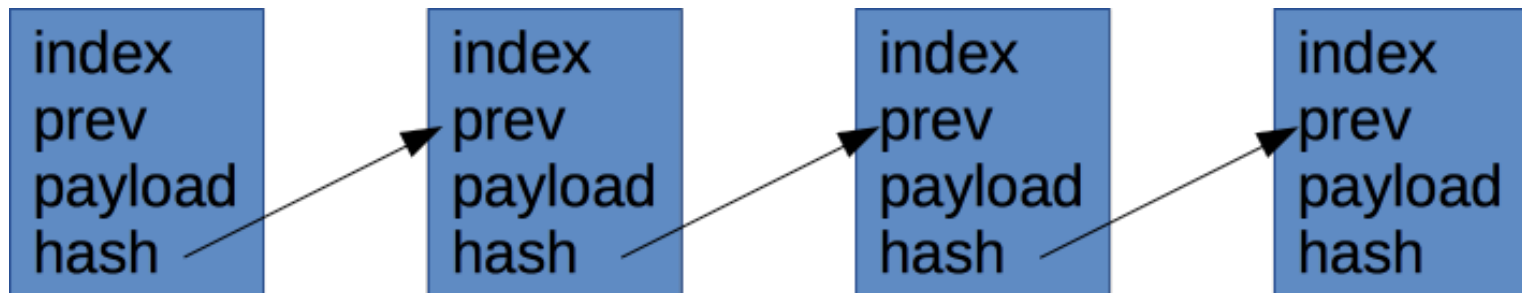


Figure: A sequence of records linked together; each record contains the hash of the previous record.

What does that buy us?

- We're using a cryptographic hash function like SHA256.
- That means nobody can find two inputs with the same hash value.
- ...and **that** means that record N contains a **commitment** to record N-1
- ...which contains a commitment to record N-2, which contains a commitment to record N-3, and so on.

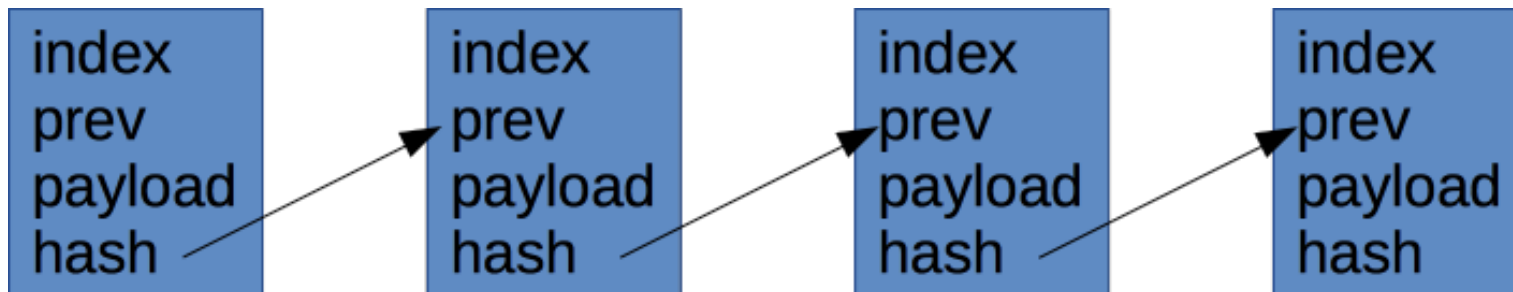


Figure: A sequence of records linked together; each record contains the hash of the previous record.

Hash chains and block chains

- Hash chains have the property that every record contains a commitment to all previous records.
 - If you change record N , this changes the final hashes of records $N+1$, $N+2$, ...
- Result: Once we all accept record N , we have locked in the contents of record 1, 2, 3, ..., $N-1$ as well.
- Blockchains use hash chains as a component
- Hash chains are also useful in a lot of other contexts
 - For example, a system with a trusted arbiter can use a hash chain to limit the arbiter's power—even the arbiter can't change history.

The block chain

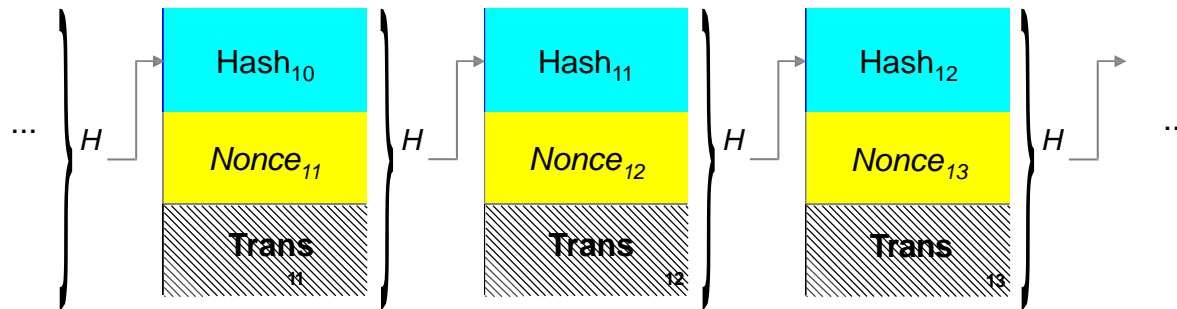


Figure: A block chain containing three blocks, each containing the hash of the previous block, and each containing a sequence of transactions and a nonce.

- Each block in the chain *commits* to all previous blocks and transactions

Building Block: Validity conditions

What will the world accept as the next block?

- We don't have some trusted entity to decide what may be added to block chain
- ...so we have to decide what blocks are valid.
- Example: Bitcoin
 - Signatures needed for moving BTC from an account
 - Not allowed to leave a negative balance in an account
 - Block must contain correct proof-of-work
- A proposed additional block that doesn't meet these conditions won't be accepted by the rest of the network.

Enforced by consensus

Forked chains

- Blockchains are used in distributed systems
 - It's possible for different parties in the system to have a disagreement
 - There's no trusted arbiter to decide who is right!
 - Any blockchain must deal with this issue somehow!
-
- Suppose Alice and Bob no longer agree on the state of the world
 - (because they don't agree on history)
 - How do they come to an agreement on who's right?
 - Several different techniques to resolve disagreements

Adding new blocks to the chain

- Any blockchain system has to determine who can add new blocks to the chain, and how it's done.
- Two main ideas I'll discuss below
 - Proof of work
 - Permissioned blockchain
- Also more ideas I'm not going to talk about
 - Proof of stake
 - Proof of storage
 - Probably several more I've never heard of

Building Block: Proof of Work

- I want you to do a big computation.
 - I want you to prove you did it.
 - I don't want to do much work checking the proof.
- Why is this useful?
 - Limits the rate of new blocks
 - Makes attempts to add invalid blocks to the chain expensive
 - Provides a clear way to decide between competing chains when there is a disagreement—the one with the most work wins.

Note: Not all blockchains use proof of work

Hash-based proof of work

- I give you challenge C and limit $L = 2^{220}$.
- Ask you to find N such that
$$SHA256(C || N) < L$$
- Expected work = 2^{36}
- Each new N has prob 2^{-36} of success
- When you succeed, only takes me one hash to check.

This is more-or-less Adam Back's hashcash scheme

Proofs of work in every block

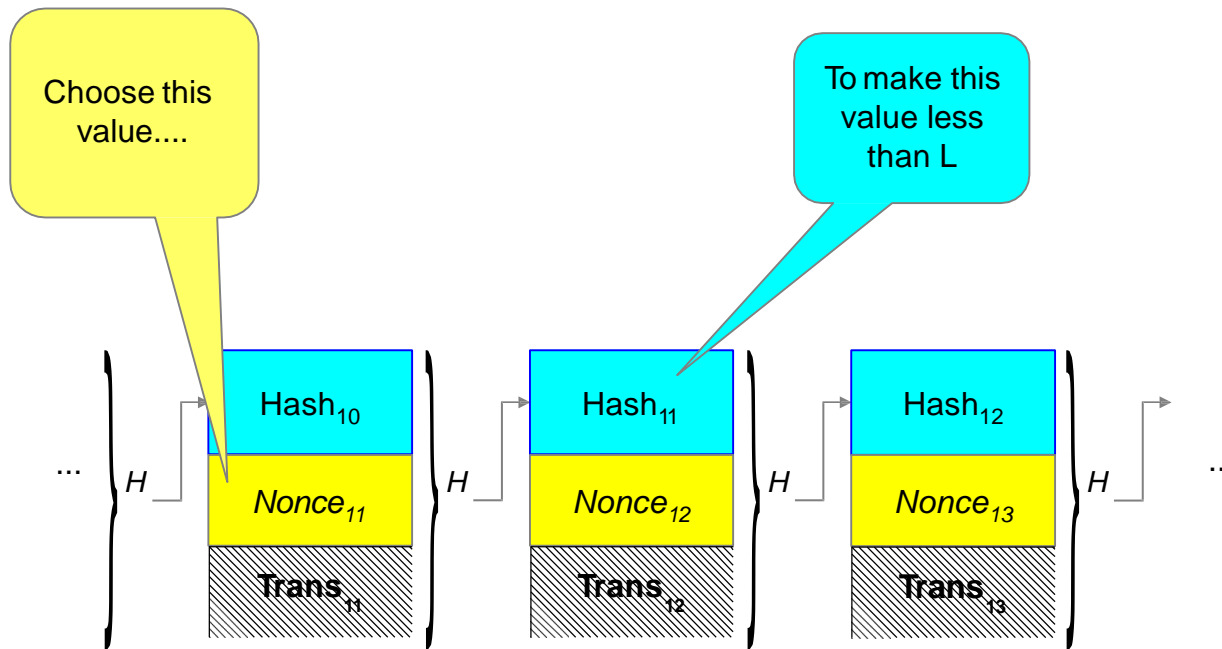


Figure: Three blocks in a block chain. The nonce in the first block is chosen to force its hash value (which appears in the second block) to be less than L.

Proofs of work solve some problems...

- We can resolve disagreements.
 - When chain forks, take fork with most work.
 - When there's a tie, keep working till one of the chains has the most work.
- Discourage people trying to add invalid blocks to chain.
 - You spend money adding a block to chain...
 - ...but if it's not valid, nobody accepts it.
- Part of how Bitcoin's very clever design of incentives works.

...but introduce others

- Expensive—lots of energy used to generate proofs
 - Done by “miners” in Bitcoin
 - Use special-purpose mining rigs optimized for doing proofs of work.
 - Environmental impact—uses lots of power, accomplishing no useful goal except keeping blockchain working
- Slow—proof of work seems to put a limit on transaction speed
 - Even more when you consider need to resolve potential disagreements
 - Bitcoin rule of thumb is wait 6 blocks (about an hour) to be sure of transaction

Permissioned blockchains

- An alternative to proof-of-work
- We have set of somewhat-trusted entities who can work together to add records to the blockchain.
- For example, we could have five trustees, and if any 3/5 vote in favor of accepting a block on the chain, then the block is added.
- Validity condition for adding a block = 3/5 signatures
- Resolution for conflicting chains = look for longest chain (aka most votes)
 - With 3/5 there shouldn't be any forked chains—someone would have to vote for two competing blocks!

Incentive design

- The real genius in Bitcoin's design is the way incentives are aligned
 - Untrusted, self-interested miners keep the system working
 - They have a big incentive to follow the protocol
 - They have substantial capital invested in Bitcoin, so they also have an incentive to avoid any attack that would undermine their investment
 - This all works because Bitcoin is all about moving money around, so it's easy to build payoffs into the protocol.
- Other blockchains (especially permissioned ones) have to find alternatives to incentives
 - Not so obvious how to build a payoff into a protocol to store medical records

Why do we trust trustees?

- Existing business or legal arrangements?
- Incentives for playing fair?
- Reputation?
- We have the “And then you go to jail” problem....

“And then you go to jail”

- It's usually a bad idea to build crypto protocols that rely on outside enforcement mechanisms
 - Sending misbehaving users to jail
 - Suing people who don't follow the protocol
 - Assuming that damage to someone's reputation will convince them to behave properly.
- Lots of examples of these things not working
 - Even if they do work, they tend to be slow
- This is something any permissioned blockchain has to solve

Wrapup 1: Blockchains let us agree on history

- We don't have to trust each other
- We don't have to have a trusted third party
- System is distributed
- Agreeing on history → agreeing on state of system

Wrapup 2: Blockchains and hash chains

- The Nth record in the hash chain commits to all previous records.
- Can't change any previous record without making hash chain invalid.
- A blockchain is a hash chain with some other stuff added
 - Validity conditions
 - Way to resolve disagreements

Wrapup 3: Permissioned vs Proof-of-work

- Most blockchains in use now use proof-of-work
- Many new proposals use permissioned blockchains
 - Some set of somewhat-trusted entities
- There are other ways to do it
 - Proof of storage
 - Proof of stake
 - Probably more I don't know about

Questions?
