

CSE446: Blockchain & Cryptocurrencies

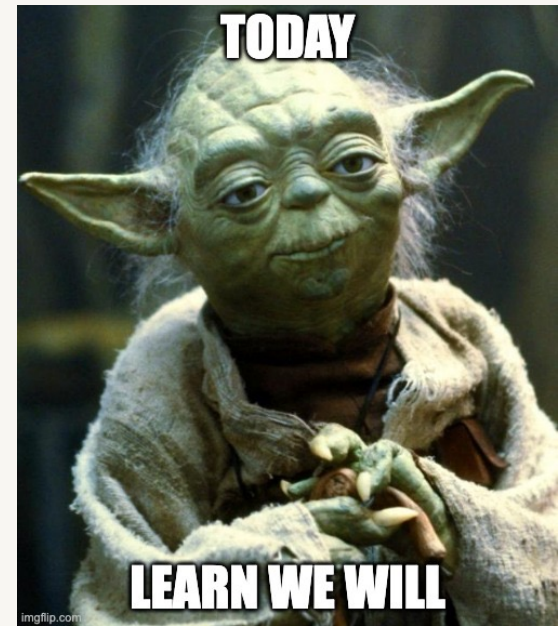
Lecture – 3: Cryptography Review



Inspiring Excellence

Agenda

- Cryptography review
 - Cryptographic hash functions
 - Symmetric encryption
 - Asymmetric encryption (Public-key encryption)
 - Digital signature
 - Merkle tree



This lecture has been prepared from multiple sources:

- Textbook
- <https://github.com/PratyushRT/blockchainsS21/wiki>
- <https://github.com/sebischair/bbse>

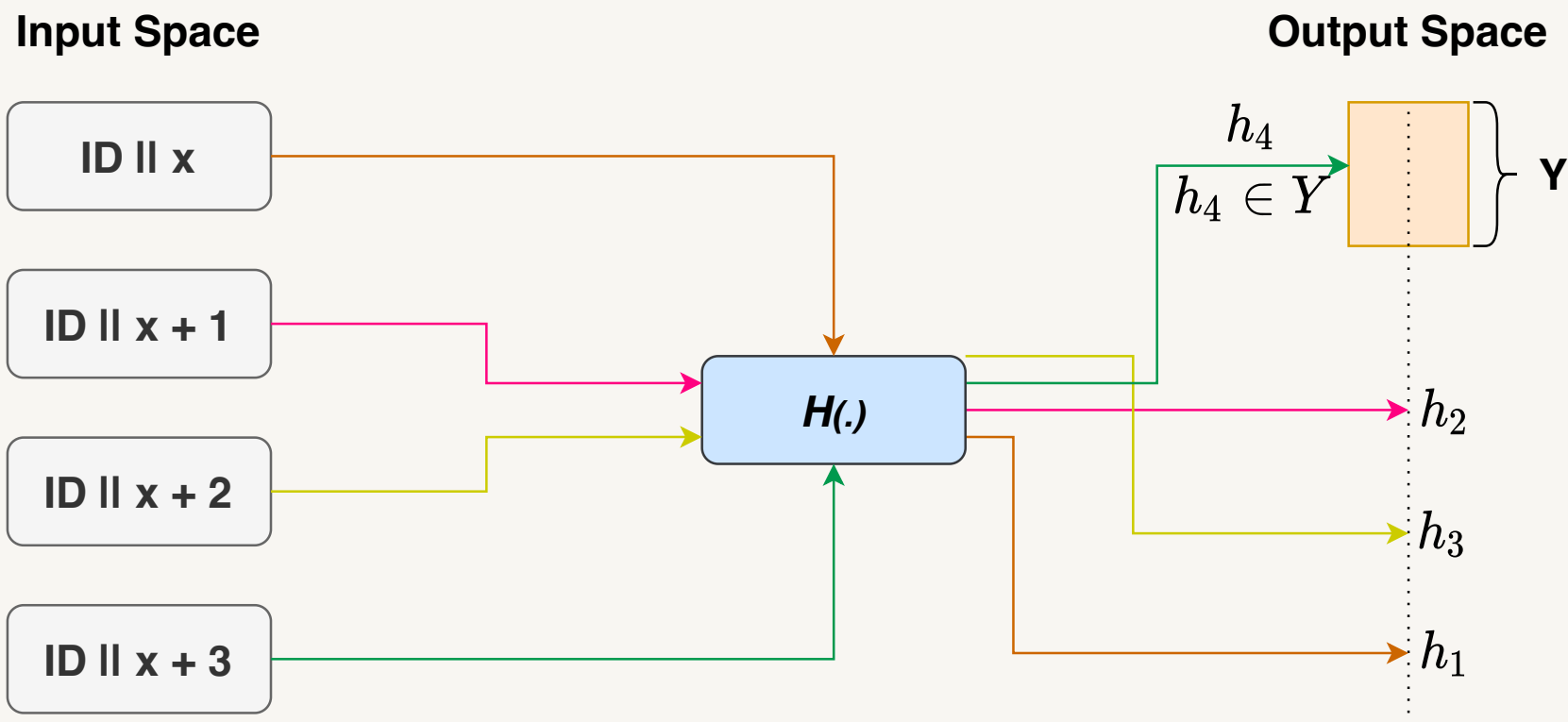
Cryptographic hash function: puzzle-friendliness

- A hash function H is said to be puzzle-friendly if
 - for every possible n -bit output value y , if k is chosen from a distribution with high min-entropy,
 - then it is infeasible to find x such that $H(k \parallel x) = y$ in time significantly less than 2^n
- If a hash function is puzzle friendly, then there is no solving strategy for this type of puzzle that is much better than trying random values of x

Puzzle~friendliness: application

- Search puzzle
- Consists out of:
 - A hash function H : Computes the *puzzle results*
 - A value id : *puzzle-ID* (makes solutions to the puzzle unique, should not be known in advance, otherwise pre-computation is possible)
 - A target set Y , for a valid solution z , $z \in Y$
 - Computation: $z = H(\text{puzzle-ID} \parallel x)$
 - x changes until $z \in Y$

Puzzle~friendliness: application



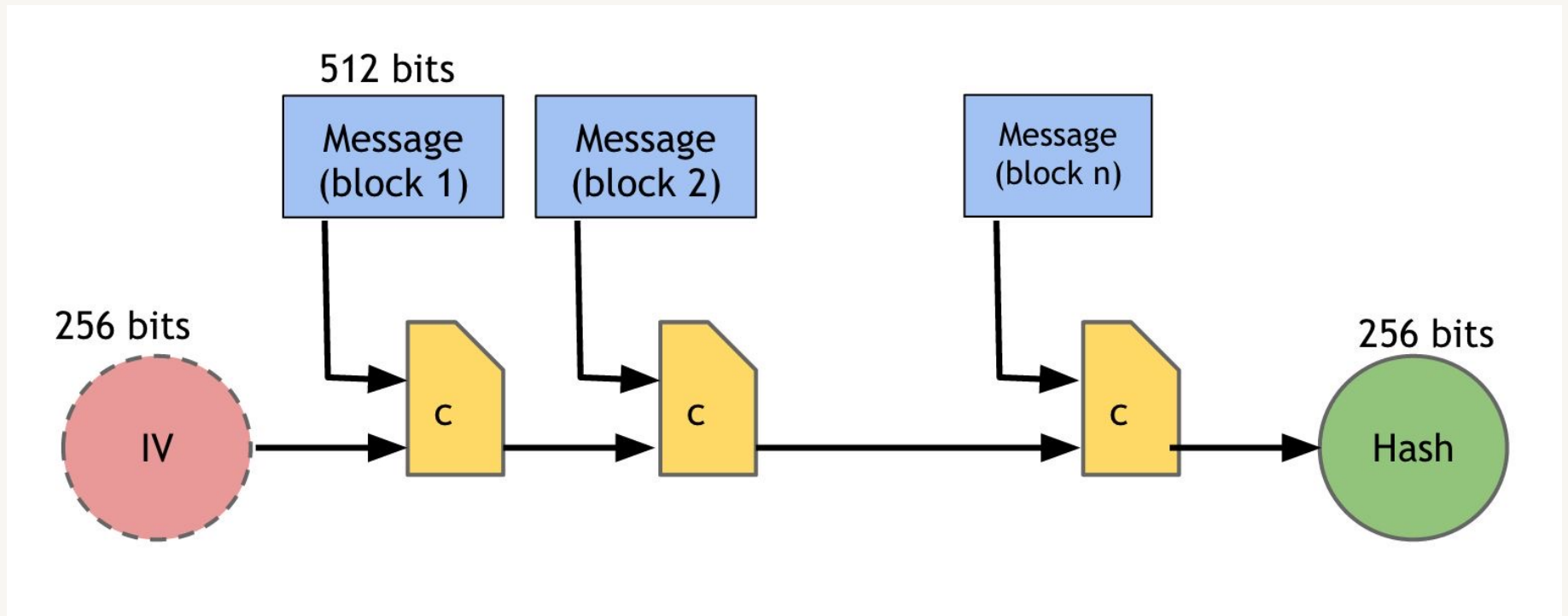
Family of hash functions

- MD5 (Message Digest 5) -> Currently considered broken!
- Secure Hashing Algorithm 1 (SHA-1) -> Currently considered broken!
- Secure Hashing Algorithm 2/3 (SHA-2/3) -> safe to use, SHA-3 preferable

SHA-256 (SHA-2)

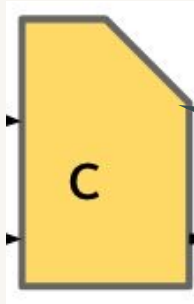
Break the message in the multiples of 512 bits, pad 0s in the last block to make it a 512 bit block

Merkle-Damgard
Construction



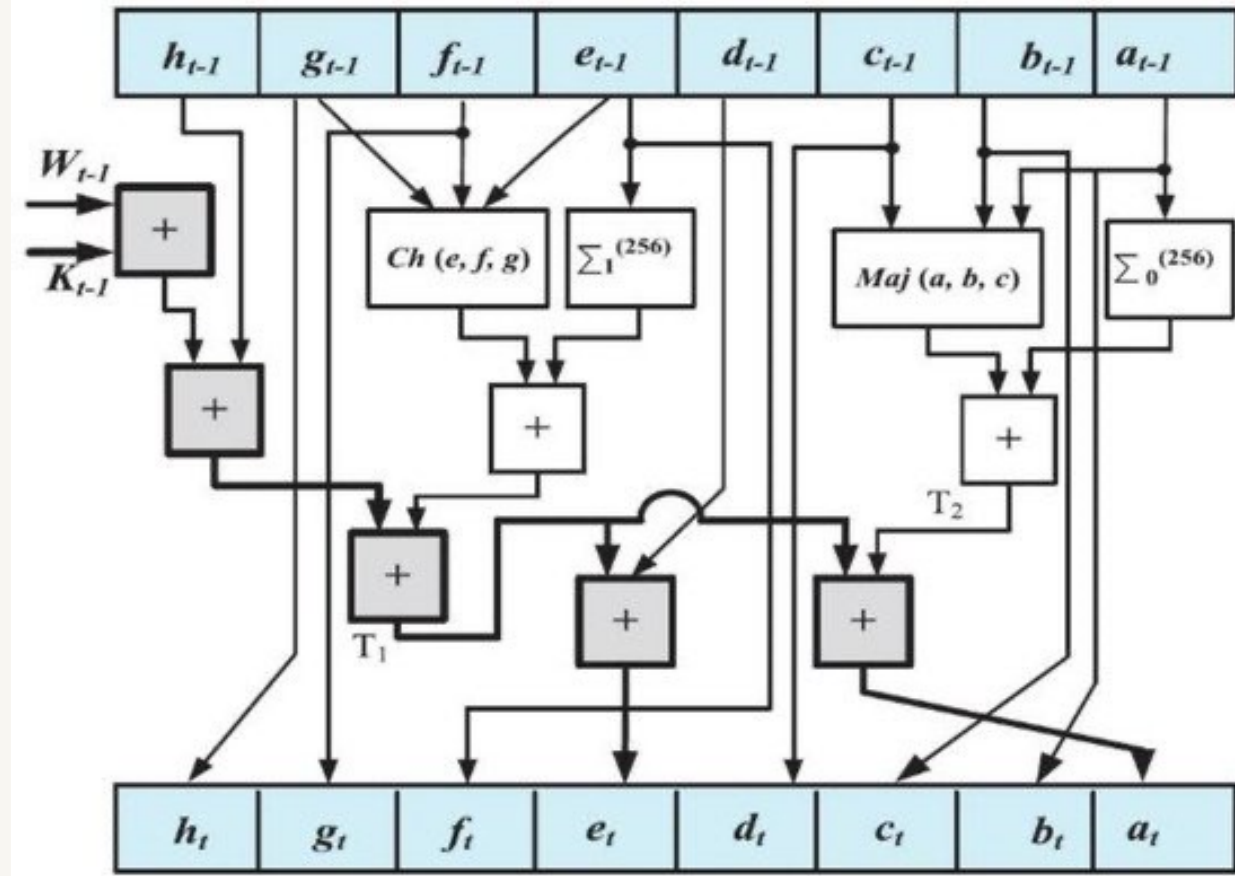
Theorem [Merkle-Damgard]: If c is collision-resistant, then SHA-256 is collision-resistant

SHA-256 (SHA-2)



Theorem [Merkle-Damgard]: If c is collision-resistant, then SHA-256 is collision-resistant

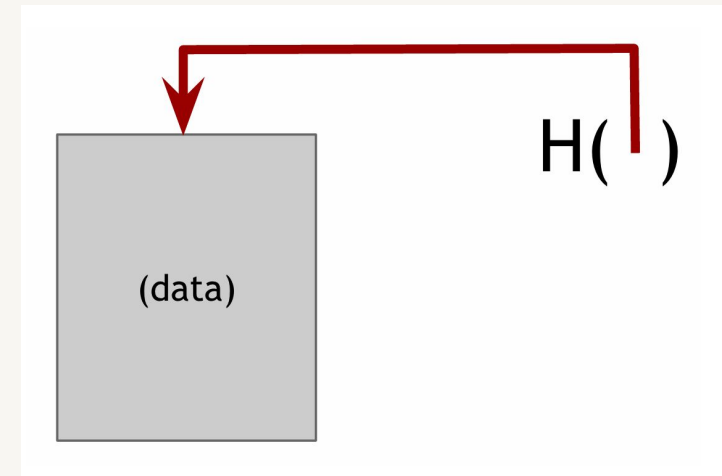
SHA-256 (SHA-2)



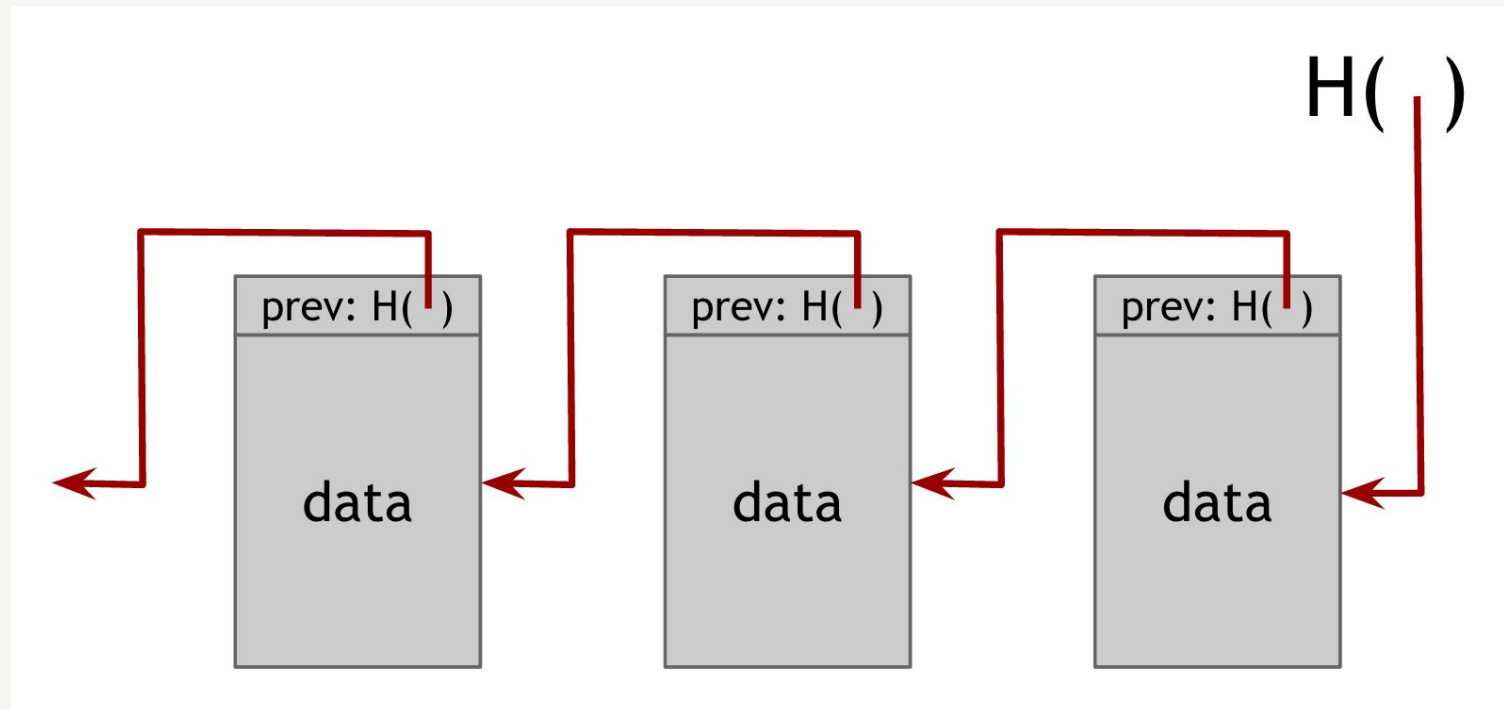
https://www.mdpi.com/entropy/entropy-21-00577/article_deploy/html/images/entropy-21-00577-g001-550.jpg

Hash pointer

- A hash pointer is
 - a pointer to where some information is stored
 - together with a cryptographic hash of the information
- We can use a hash pointer
 - to get the info back, and
 - to verify that it hasn't changed

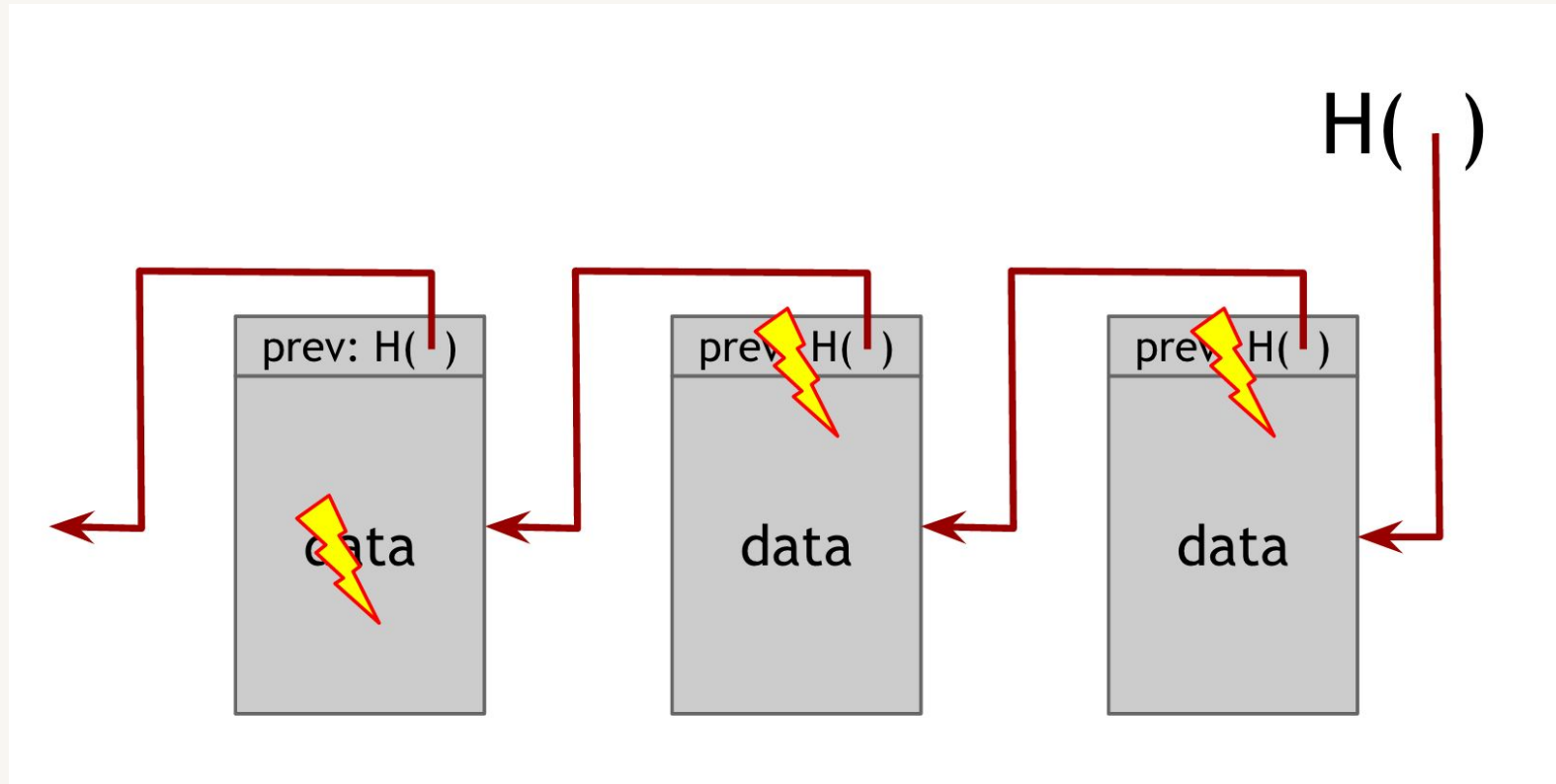


Linked list with hash pointer

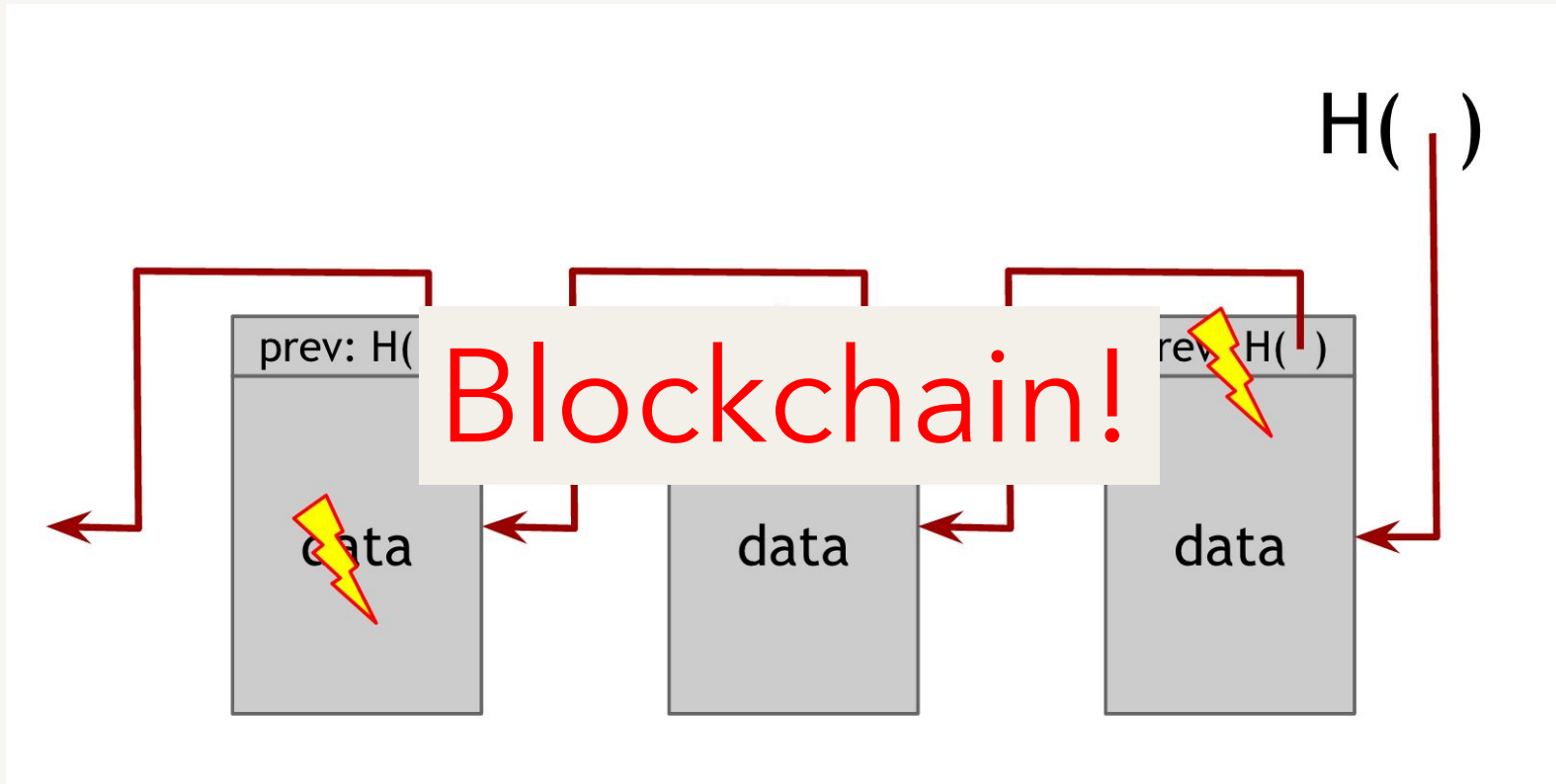


Temper-evident log

Linked list with hash pointer



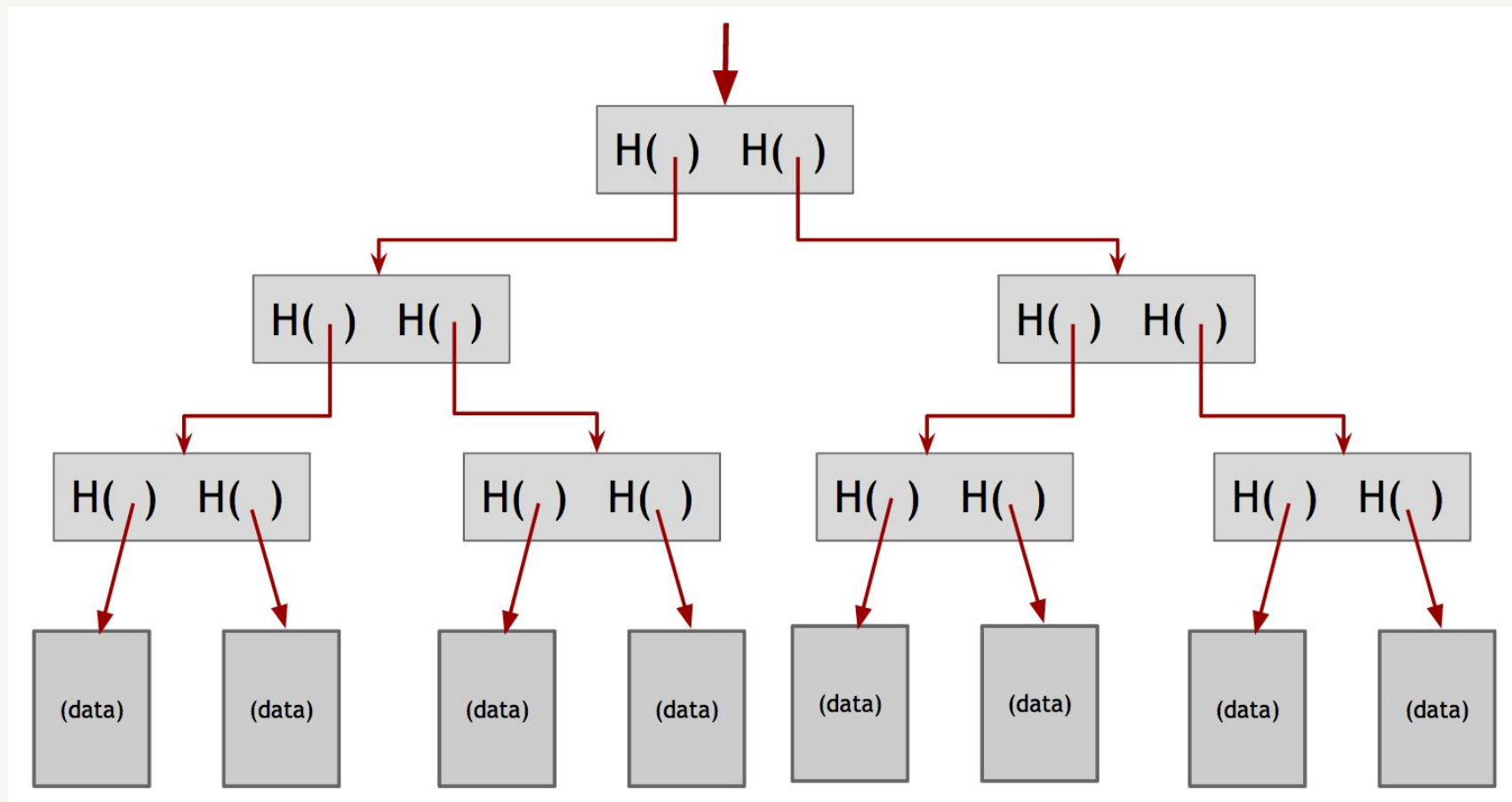
Linked list with hash pointer



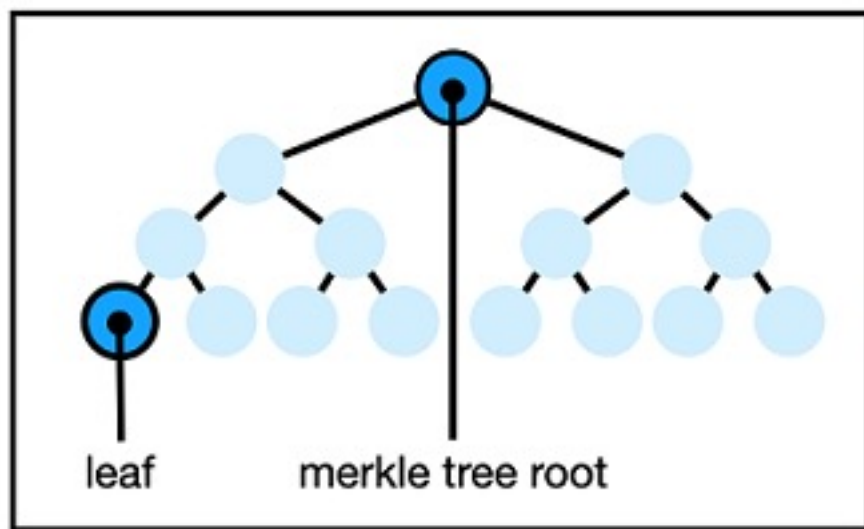
Merkle tree

- A Merkle Tree (due to Ralph Merkle) is a data structure using cryptographic hashes, basically a binary tree with hash pointers
- It is used as an efficient and secure way to verify large data structures
- It especially provides an efficient way to
 - prove that a certain data block is contained in a Merkle Tree (Proof of Membership) in $O(\log n)$ time/space
 - prove that a certain data block is not contained in a sorted Merkle Tree (Proof of Non Membership) in $O(\log n)$ time/space

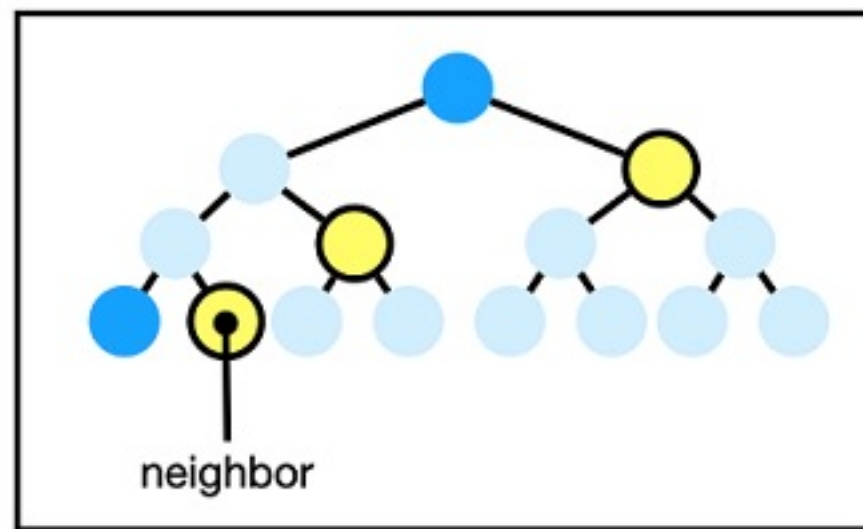
Merkle tree



Merkle tree (proof of membership)



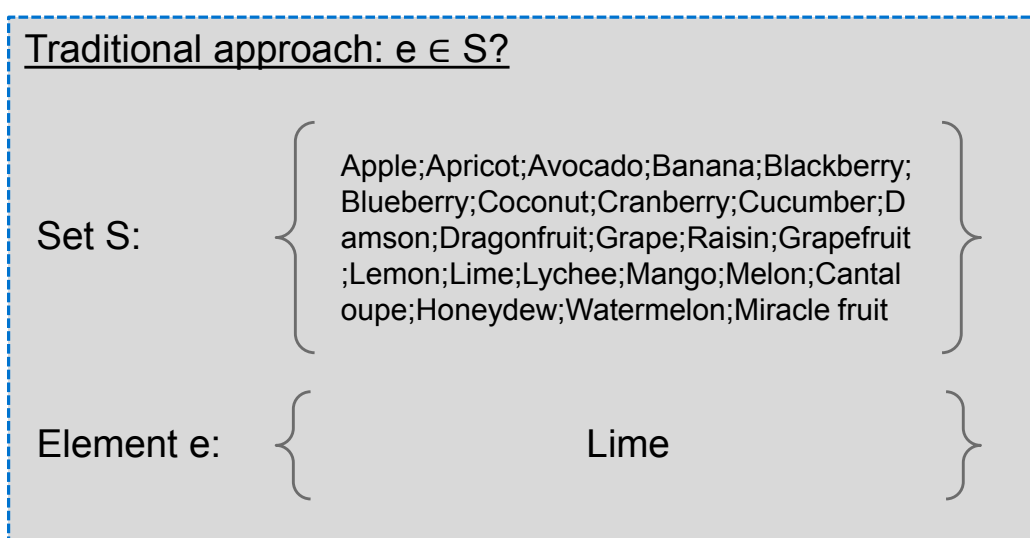
1. **proof of membership** for the **leaf**
with knowledge of the **root**



2. using the **neighbor nodes**
verify if it leads to the root

Bloom filter

- A Bloom filter is a probabilistic data structure which allows to test if an element is a member of a set
- How would you check if an element is a member of a set?



```
function isElem (_elem) {  
  foreach(elem in set) {  
    if(elem == _elem) {  
      return true;  
    }  
  } return false;  
}
```

→ If the set grows larger, membership checks for multiple items can be time-consuming

Bloom filter

You are the web developer for the website of a news paper. At the bottom of each article, the website displays recommended articles to read for the user. The algorithm for good recommendations works fine already, however does not take into account if an article has been read yet. If it has, it should not be displayed.

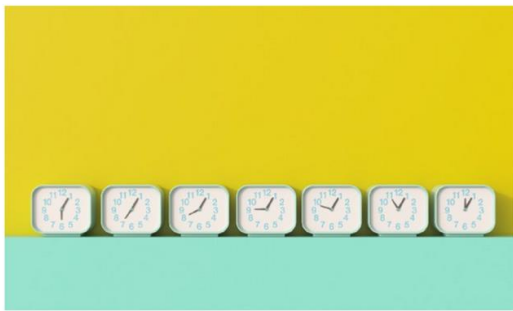
With thousands individual users each month which read many stories over time, storing all reads and testing if an article is in this set is too slow for a good surfing experience on the website of the newspaper.


READ NEXT

There's Only One Thing To Do With Today: Seize It

Understanding carpe diem is easy. Living it is harder.

Ryan Holiday · Feb 14 · 10 min read ★






Facebook's Users Are More Powerless Than Ever

We live in a world where sharing is inescapable. We're being used, but we have no other option.


Colin Horgan · Feb 14 · 6 min read ★



Reflecting on My Failure to Build a Billion-Dollar Company

It took years for me to realize my pursuit of growth was misguided from the start

Sahil Lavingia · Feb 7 · 13 min read ★



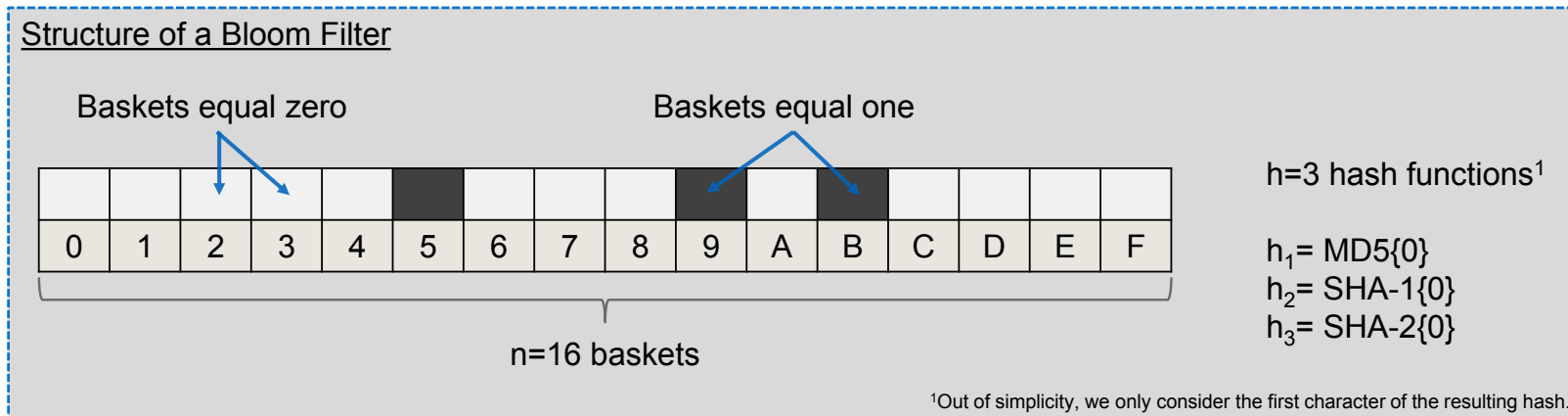
What Poverty Taught Me About Being 'Too Generous'

Using the hunger I experienced as a kid to teach mine the power of generosity

Kristine Levine · Jan 17 · 6 min read ★

Bloom filter

- A Bloom filter is a probabilistic data structure which allows to test if $e \in S$
 - It is set up as a bit array
- A query to the filter either returns
 - True ("e possibly in set S")
 - False ("e definitely not in set S")



Bloom filter: phase 0

Set Information

Set S: { empty }

Element e: empty

Hashing results

$h_1 =$

$h_2 =$

$h_3 =$

$h_1 = \text{MD5}\{0\}$
 $h_2 = \text{SHA-1}\{0\}$
 $h_3 = \text{SHA-2}\{0\}$

Explicit State

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Phase 0: Filter Setup

The filter is initialized with n buckets ($0 \rightarrow n-1$). Each bucket is filled with zero. The hash-functions are defined.

This box explains every step.

Bloom filter: phase 1

Set Information

Set S: { *apple* }

Element e: *apple*

Hashing results

$h_1 = 1F3870BE274F6...$
 $h_2 = D0BE2DC421BE...$
 $h_3 = 3A7BD3E2360A...$

$h_1 = MD5\{0\}$
 $h_2 = SHA-1\{0\}$
 $h_3 = SHA-2\{0\}$

Explicit State

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Phase 1: Element addition (e="apple")

We add our first element to the filter. We hash it with three different hash functions (h_1 , h_2 , h_3) and set their corresponding buckets to one. Following buckets are set to one: 1, 3, and D.

Bloom filter: phase 1

Set Information

Set S: { *apple; lime* }

Element e: *lime*

Hashing results

$h_1 = 6$

$h_2 = C$

$h_3 = E$

$h_1 = \text{MD5}\{0\}$
 $h_2 = \text{SHA-1}\{0\}$
 $h_3 = \text{SHA-2}\{0\}$

Explicit State

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Phase 1: Element addition (e="lime")

We add our second element to the filter. We hash it with three different hash functions (h_1 , h_2 , h_3) and set their corresponding buckets to one. Following buckets are set to one: 6, C, and E.

Bloom filter: phase 1

Set Information

Set S:

apple; lime; lemon

Element e:

lemon

Hashing results

$h_1 = 3$

$h_2 = D$

$h_3 = F$

$h_1 = \text{MD5}\{0\}$
 $h_2 = \text{SHA-1}\{0\}$
 $h_3 = \text{SHA-2}\{0\}$

Explicit State

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Phase 1: Element addition (e="lemon")

We add our third element to the filter. Again, we hash it. This time, two out of three buckets are already set to one. In these buckets one remains.

Bloom filter: phase 2

Set Information

Set S: { *apple; lime; lemon* }

Element search

apple

Hashing results

$h_1 = 1$
 $h_2 = D$
 $h_3 = 3$

$h_1 = \text{MD5}\{0\}$
 $h_2 = \text{SHA-1}\{0\}$
 $h_3 = \text{SHA-2}\{0\}$

Explicit State

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Phase 2: Element Validation (e="apple")

Now we are able to search if an element is contained in the bloom filter. We added the apple before, therefore we receive *true* from the filter.

Bloom filter: phase 2

Set Information

Set S:

apple; lime; lemon

Element search

mango

Hashing results

$h_1 = A$

$h_2 = 9$

$h_3 = 6$

$h_1 = \text{MD5}\{0\}$
 $h_2 = \text{SHA-1}\{0\}$
 $h_3 = \text{SHA-2}\{0\}$

Explicit State

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Phase 2: Element Validation (e="mango")

Is "mango" contained in the set? Hashing mango results in A, 9, and 6, of which only one bucket is set to one.

All h buckets have to be set to one for a match.

Bloom filter: phase 2

Set Information

Set S:

apple; lime; lemon

Element search

grapefruit

Hashing results

$h_1 = 6$

$h_2 = F$

$h_3 = E$

$h_1 = \text{MD5}\{0\}$
 $h_2 = \text{SHA-1}\{0\}$
 $h_3 = \text{SHA-2}\{0\}$

Explicit State

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Phase 2: Element Validation (e="grapefruit")

Other elements might generate a false positive, as their buckets might be filled by chance. This is the case with grapefruit, as the functions generate the hashes 6, F, and E.

False positive generations occur in Bloom filters. The probability of their occurrence depends on the number of buckets n and the number of hash functions h .


Bloom filter


READ NEXT

There's Only One Thing To Do With Today: Seize It

Understanding carpe diem is easy. Living it is harder.

Ryan Holiday · Feb 14 · 10 min read ★






Facebook's Users Are More Powerless Than Ever

We live in a world where sharing is inescapable. We're being used, but we have no other option.


Colin Horgan · Feb 14 · 6 min read ★



Reflecting on My Failure to Build a Billion-Dollar Company

It took years for me to realize my pursuit of growth was misguided from the start

Sahil Lavingia · Feb 7 · 13 min read ★



What Poverty Taught Me About Being 'Too Generous'

Using the hunger I experienced as a kid to teach mine the power of generosity

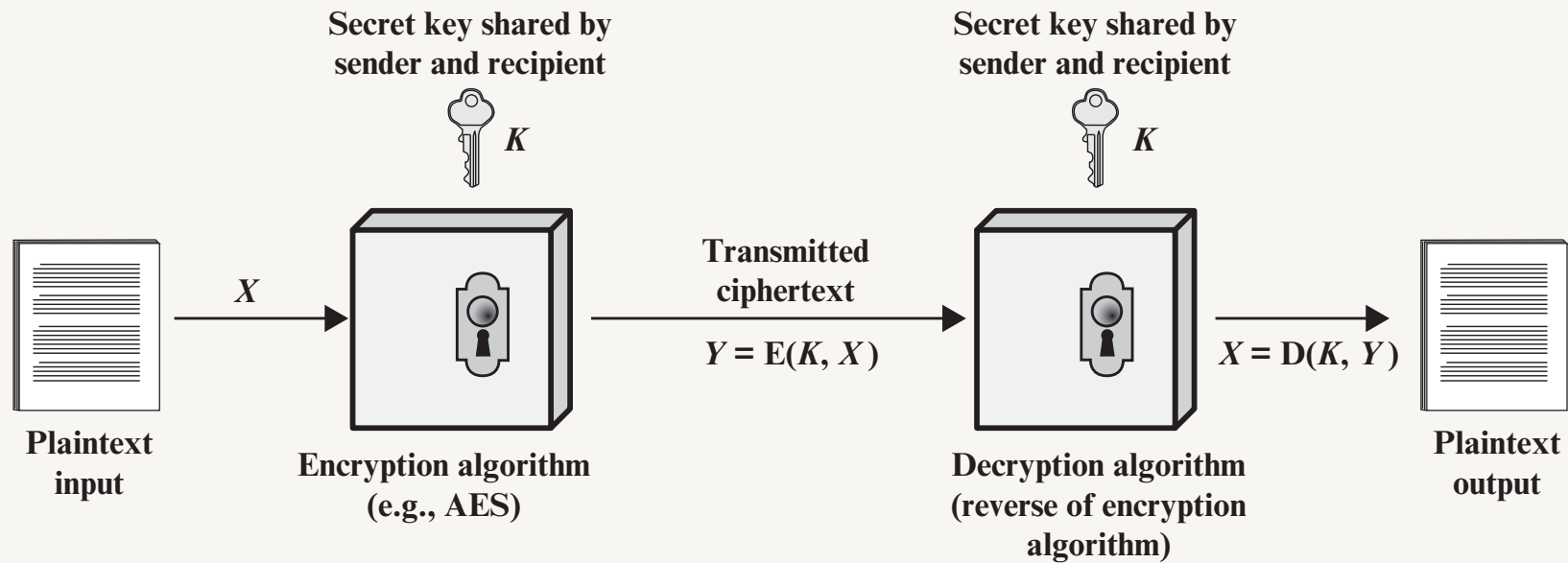
Kristine Levine · Jan 17 · 6 min read ★

What to do?

- Store a bloom filter for each user.
- Add each read story to the filter.
- Check the filter to find out if a story has been read:
 - True: Do not display story
 - False: Display story and store in filter

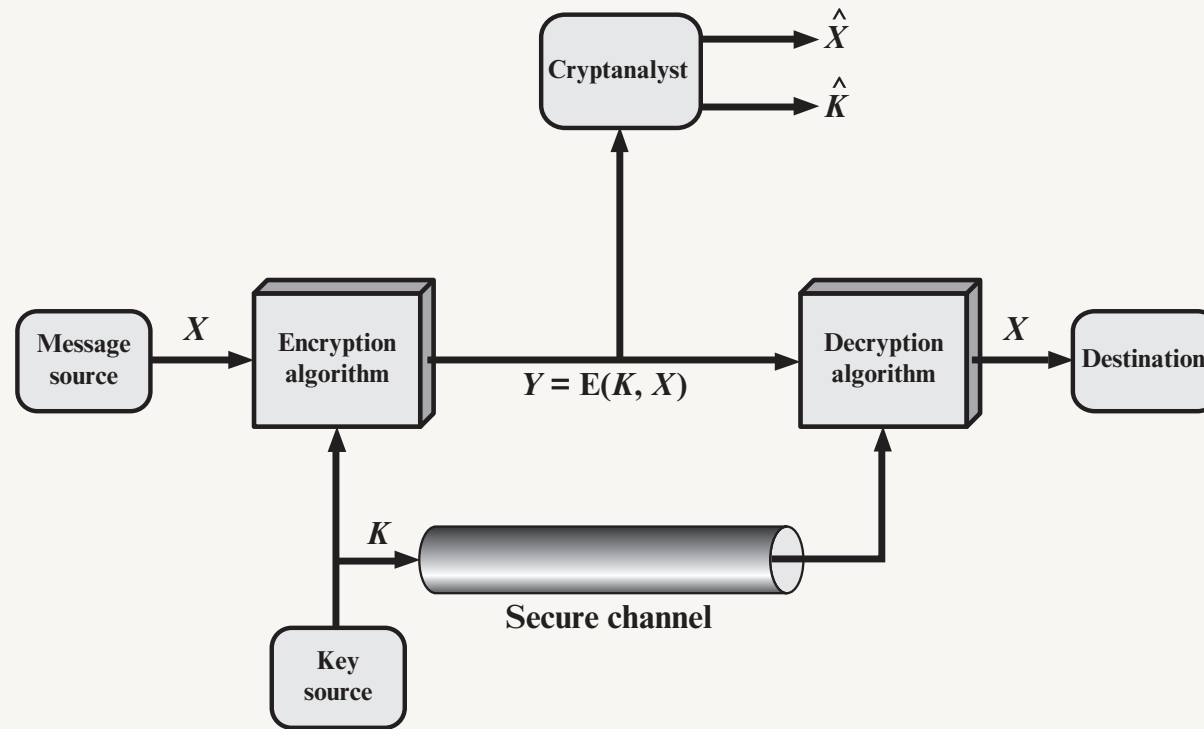
Why are false positives not a problem in this case?

Symmetric encryption



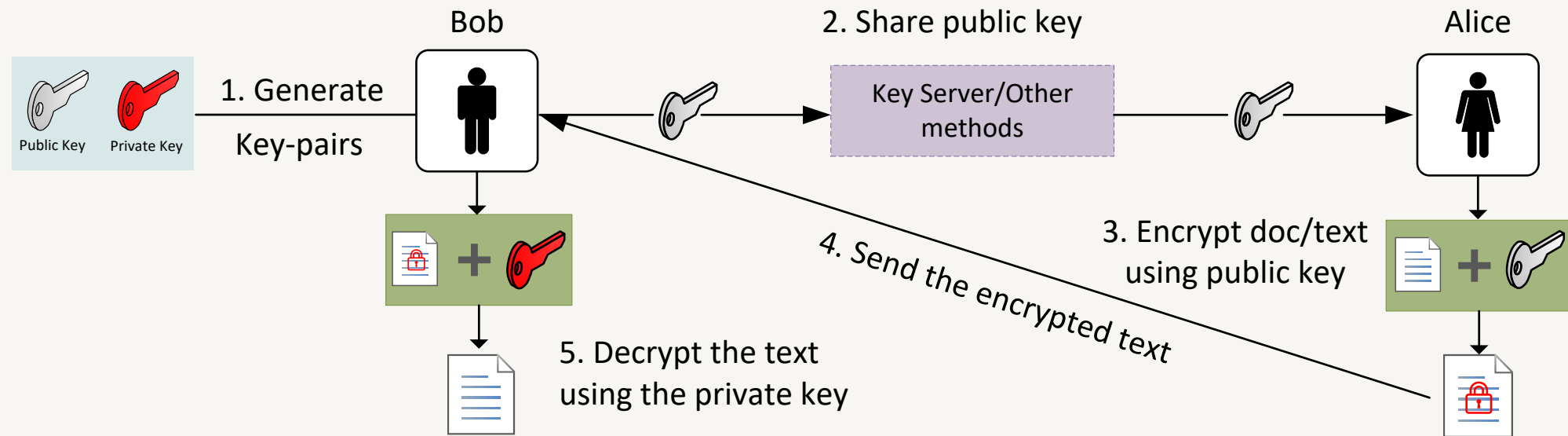
The main issue is key-management!

Symmetric encryption

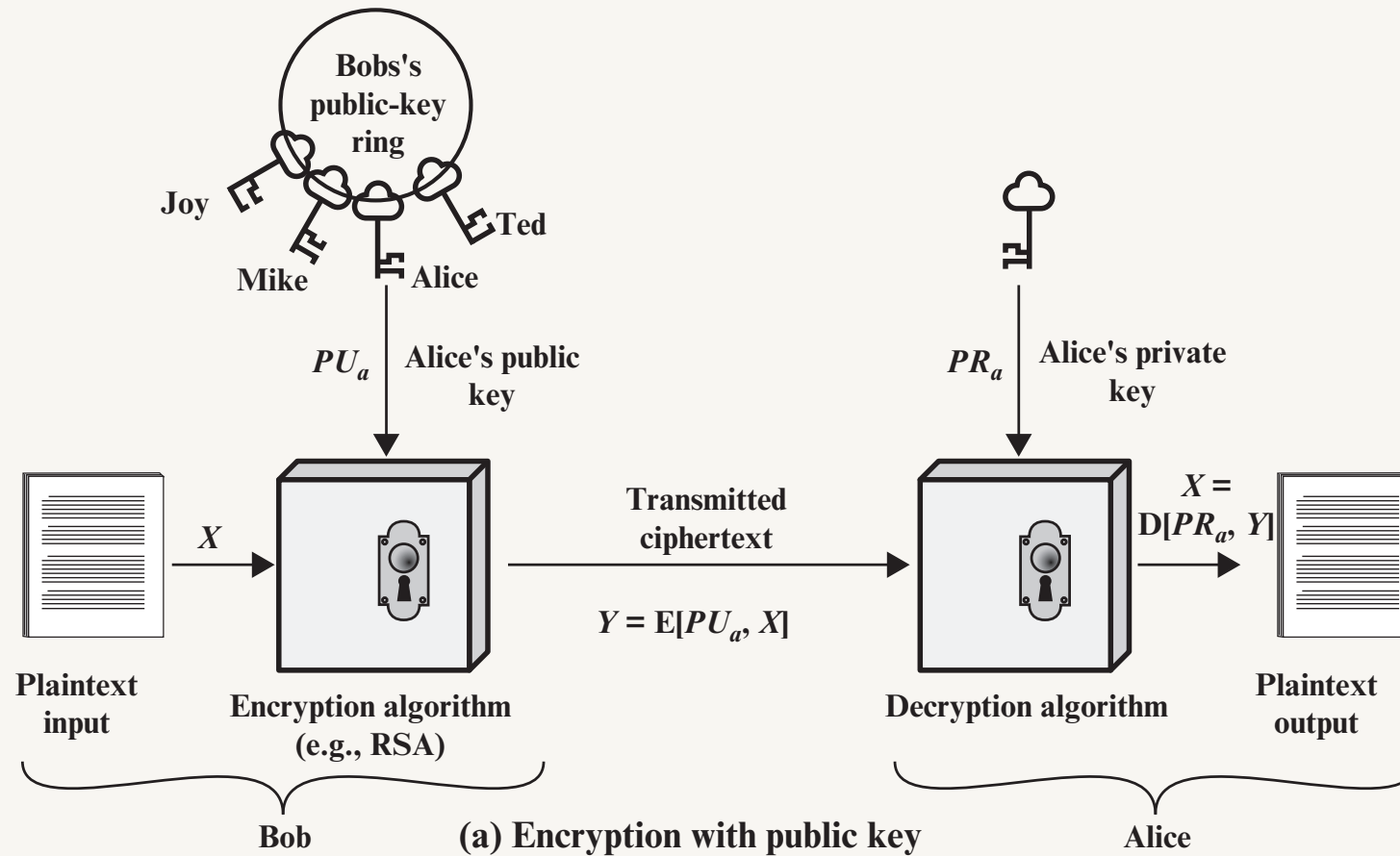


AES is currently the most widely used symmetric encryption

Public key cryptography: encryption



Public key cryptography: encryption



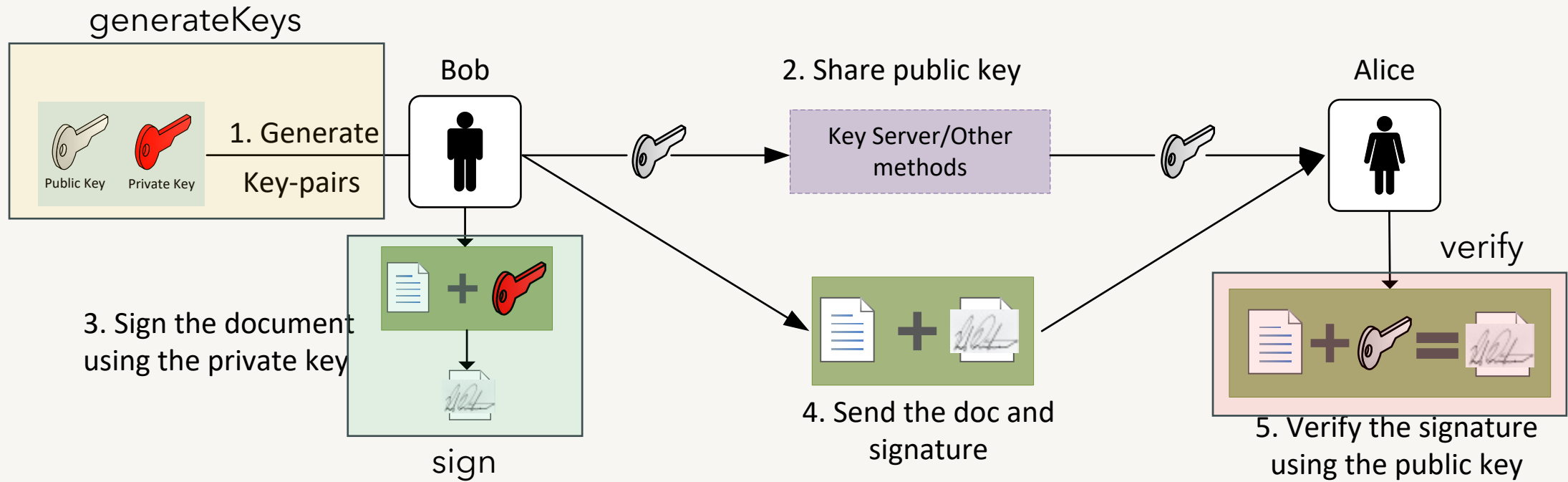
Public key cryptography: digital signature

- Digital signatures are based on asymmetric cryptography algorithms like RSA or ECC
- We need two properties of (analogue) signatures to hold in the digital world:
 - Only an entity is able to create a signature of its own, but everyone can verify it
 - This signature is tied to data that gets signed. A signature cannot be used for different data

Digital signature: definition

- Three algorithms:
- $(sk, pk) := \text{generateKeys}(\text{keysize})$
 - sk is the secret key and is used to sign messages. pk is the public key and is given to everyone. With the pk , they can verify the signature
- $\text{sig} := \text{sign}(sk, \text{message})$
 - The sign method takes the message and the secret key, sk , as input and returns a signature for message under sk
- $\text{isValid} := \text{verify}(pk, \text{message}, \text{sig})$
 - The verify method takes a message, a signature, and a public key as input
 - It will return true if the signature was generated out of the message and the secret key, otherwise false
- Such that $\text{verify}(pk, \text{message}, \text{sign}(sk, \text{message})) == \text{true}$ and signatures are unforgeable

Digital signature: definition



Digital signature algorithms

- Two major digital signature schemes are available
- RSA-based signature schemes, such as RSA-PSS
 - RSA signature was invented 1977 by Rivest, Shamir and Adleman
 - Based on the assumption that the factorisation of large prime number multiplied is very hard
- ECC-based signature schemes, such as ECDSA
 - Suggested independently by Neal Koblitz and Victor S. Miller in 1985
 - Based on discrete logarithms
- The BSI recommends following key sizes for asymmetric cryptography
 - RSA: min. 2048 Bit
 - ECDSA: min. 256 Bit
- Due to smaller key sizes in ECC, many blockchain systems use ECC

Cryptography resources

- Introduction to Cryptography: With Coding Theory, by Lawrence C. Washington and Wade Trappe
- Cryptography and Network Security: Principles and Practice, by William Stallings



ANY QUESTION?