HASH FUNCTIONS

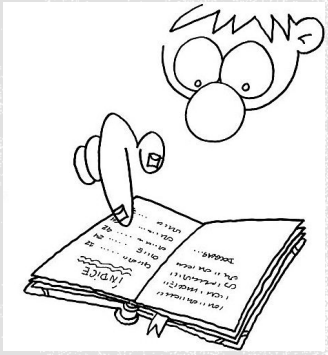
Applied Cryptography

Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

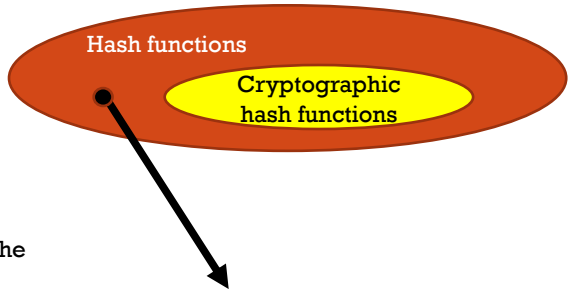
UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

CONTENTS

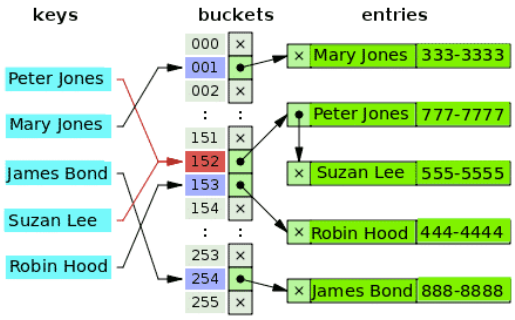


INTRODUCTION (1)

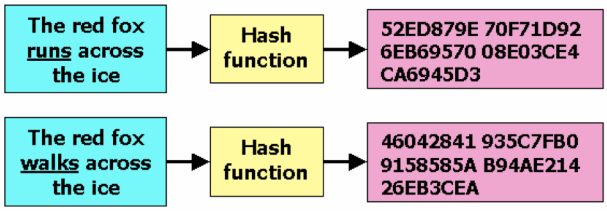


- A hash function takes a variable sized input message and produces a fixed-sized output
  - The output is referred to as the hash-code or the hash value or the message digest

- Every cryptographic hash function is a hash function
- Not every hash function is a cryptographic hash function
- A cryptographic hash function aims to guarantee a number of security properties
- Non-cryptographic hash functions try to avoid collisions for (non-malicious) inputs



INTRODUCTION (2)



- **Example of cryptographic hash function**
  - The SHA-512 hash function takes as input messages of length up to 2,128 bits and produces as output a 512-bit message digest
  - SHA stands for Secure Hash Algorithm
- **The digest can be seen as a fixed-sized fingerprint of a variable-sized message**
- Message digests produced by the most commonly used hash functions range in length from 160 to 512 bits depending on the algorithm used
- Since **a message digest depends on all the bits in the input message, any alteration of the input message during transmission would cause its message digest to not match with its original message digest**
  - This property can be used to check for forgeries, unauthorized alterations, …

4

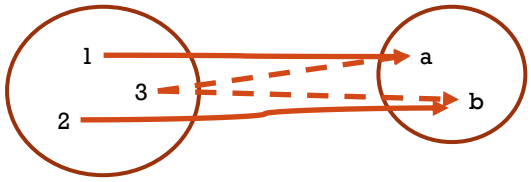
CHARACTERIZING CRYPTOGRAPHIC HASH FUNCTIONS

DEFINITION

- A hash function is called **cryptographically secure** if the following two conditions are satisfied: it is computationally infeasible to find
  - *a message that corresponds to a given digest*
    - This is called the **one-way** property of a hash function
    - **Remark**: a hash function must possess this property regardless of the length of the messages
      - It should be just as difficult to recover from its digest a message that is as short as, a single byte as a message that consists of millions of bytes
  - *two different messages that hash to the same digest*
    - This is called the **strong collision resistance** property of a hash function
- A weaker form of the strong collision resistance property is that for a given message, there should not correspond another message with the same digest
  - This is called the **weak collision resistance** property of a hash function

NOTE ON COLLISION RESISTANCE

- If the digest produced by a hash function consists of *n* bits, then there are only *2^n* distinct digests
- If we place no constraints on the messages and if there can be an arbitrary number of different possible messages, then obviously there will exist multiple messages giving rise to the same digest
- Collision resistance refers to the **likelihood** that two different messages will result in the same digest



7

AN APPLICATION OF HASH FUNCTIONS

MESSAGE AUTHENTICATION

- There exist several applications where confidentiality of the message content is not an issue (e.g., such as in the dissemination of popular media content) or it is less important (e.g., in-vehicle messages of modern cars) but authentication is
- Authentication implies that the message has not been altered in any way
  - I.e. it is the authentic original message as produced by its sender
- In such applications, unencrypted plaintext messages are sent along with their encrypted digests
  - Notice that this eliminates the computational overhead of encryption and decryption for the main message content and yet allow for its authentication; in real-time applications (such as in-vehicle messages) this is quite a desirable property
- Indeed, this works only if the hashing function has perfect collision resistance
- Otherwise (i.e. in case hash functions have poor collision resistance), an adversary could compute the digest of the message content and replace it with some other content that has the same digest

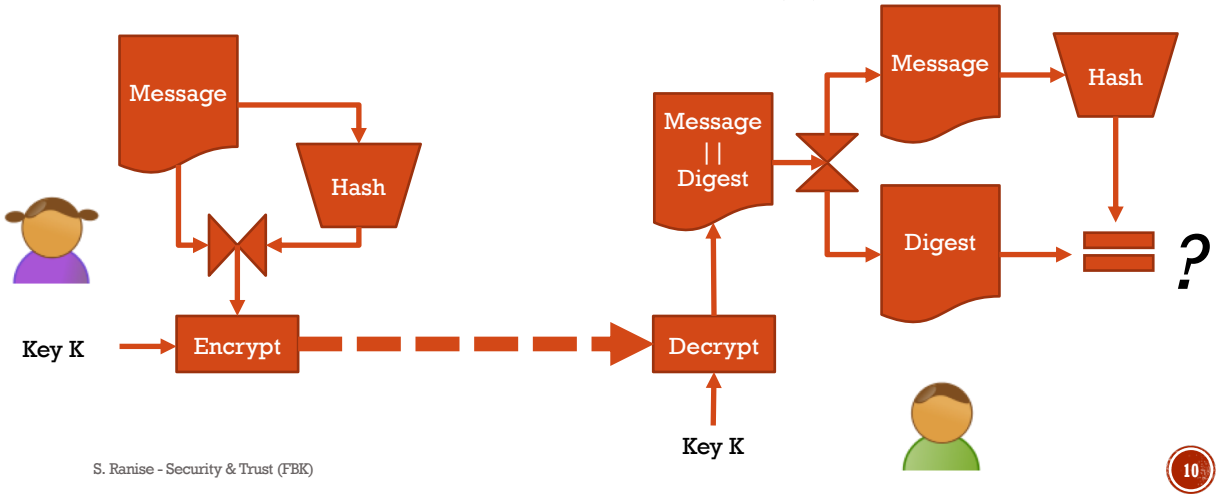
AN EXAMPLE OF THE IMPORTANCE OF INTEGRITY CHECKS IN MODERN CARS

https://www.wired.com/story/tesla-model-x-hack-bluetooth/

https://www.youtube.com/watch?v=clrNuBb3myE

Stealing a Tesla Model X because signatures have are not being checked!

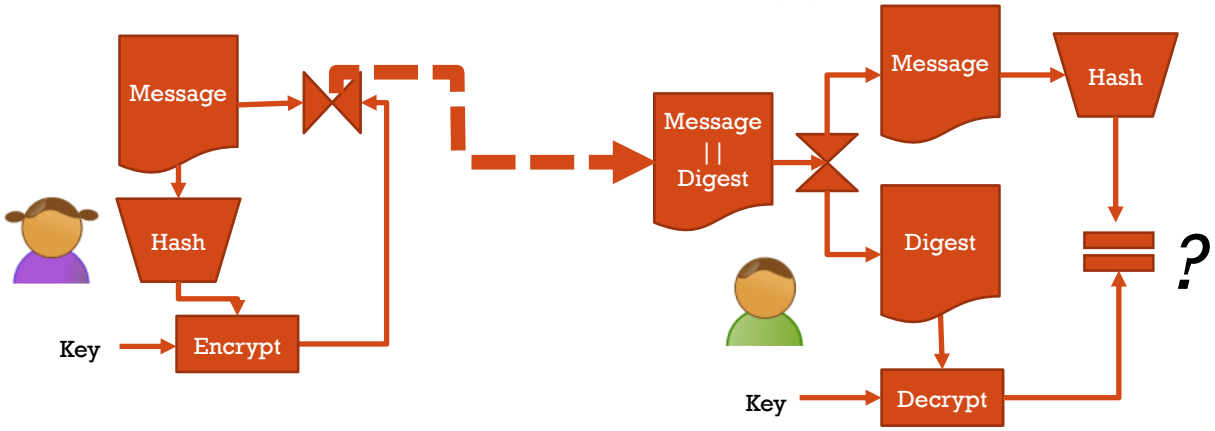
HASH FUNCTIONS FOR MESSAGE AUTHENTICATION (1)



INTERPRETATION OF THE SCHEMA

- **We assume that symmetric key cryptography is used**
  - Both Alice and Bob use the same shared key K
  - Alice concatenates the message and its digest to form a composite message that is then encrypted and sent over the network
  - Bob decrypts the message and separates out its digest, which is then compared with the digest calculated from the received message
  - The digest provides authentication and the encryption provides confidentiality

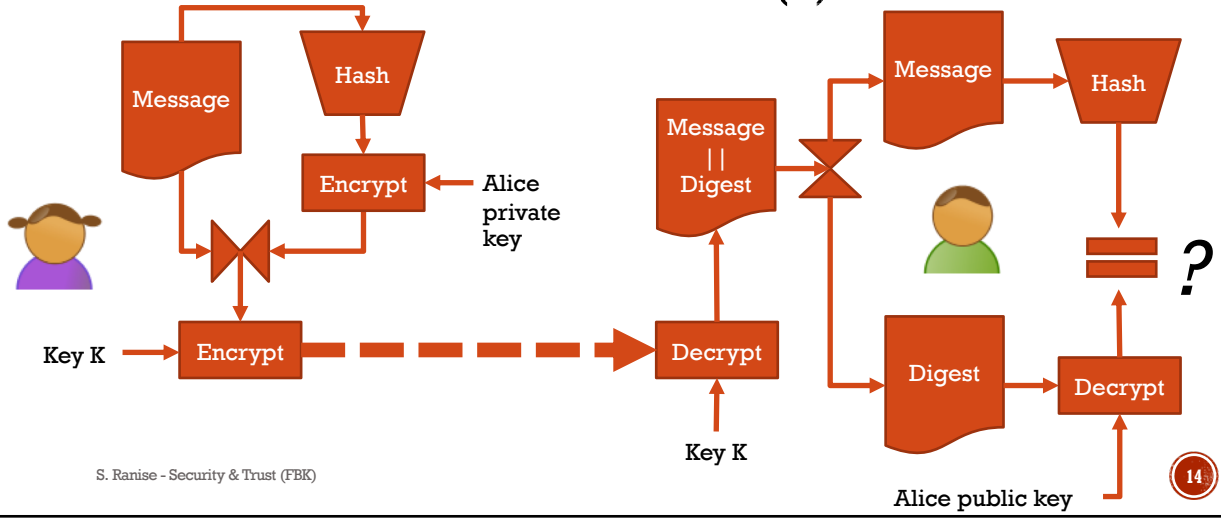
HASH FUNCTIONS FOR MESSAGE AUTHENTICATION (2)



TWO INTERPRETATIONS OF THE SCHEMA

- **Interpretation 1: symmetric key cryptography is used**
  - Variant of previous schema where only the digest is encrypted
  - This scheme is efficient to use when confidentiality is not an issue but message authentication is critical
  - Only the receiver with access to the secret key knows the real digest for the message and can verify whether or not the message is authentic
  - A digest produced in this way is known as the **Message Authentication Code (MAC)** and the hash function producing it as a **keyed hash function** (more on this below)
- **Interpretation 2: public key cryptography is used**
  - Alice uses its private key while Bob uses Alice public key
  - Confidentiality is not considered
  - The sender encrypting with its private key the digest of the message constitutes the basic idea of **digital signatures**, as explained in previous lectures

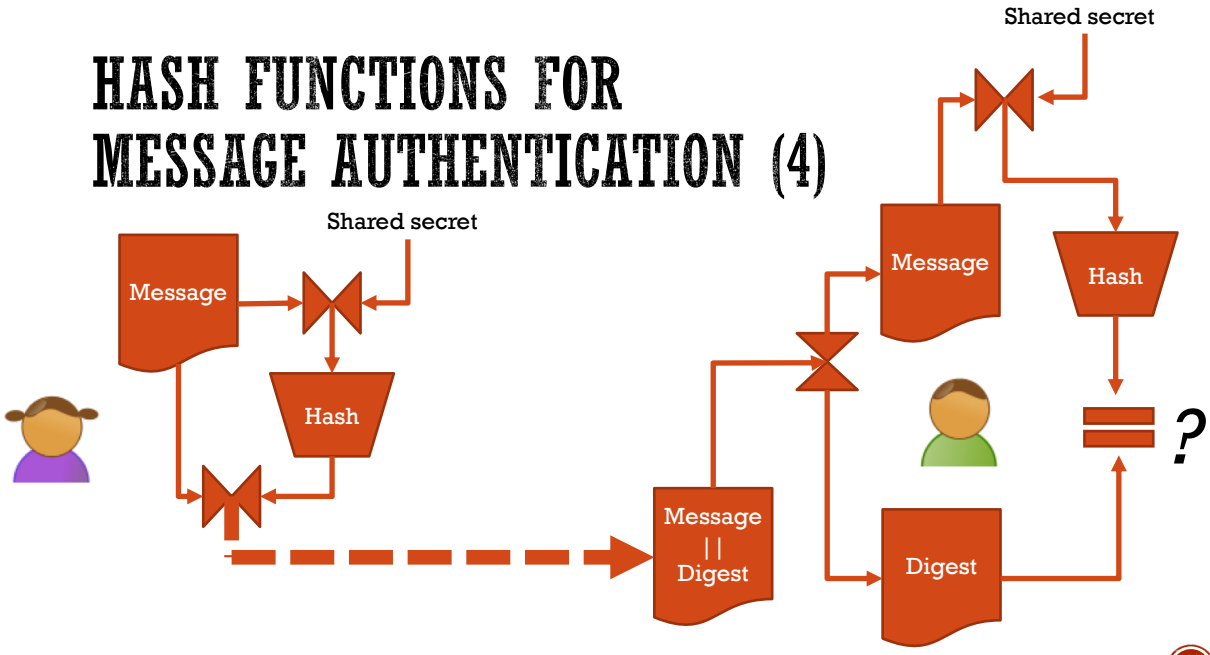
HASH FUNCTIONS FOR MESSAGE AUTHENTICATION (3)



INTERPRETATION OF THE SCHEMA

- **We assume the availability of both public and symmetric key cryptography**
  - Commonly used approach when both confidentiality and authentication are important
  - Alice encrypts the digest of the message with its private key, concatenates it with the message and then encrypts the result with the shared key
  - Bob decrypts the received message with the shared key, calculates the digest from the extracted message and compares it with the decrypted digest by using Alice public key

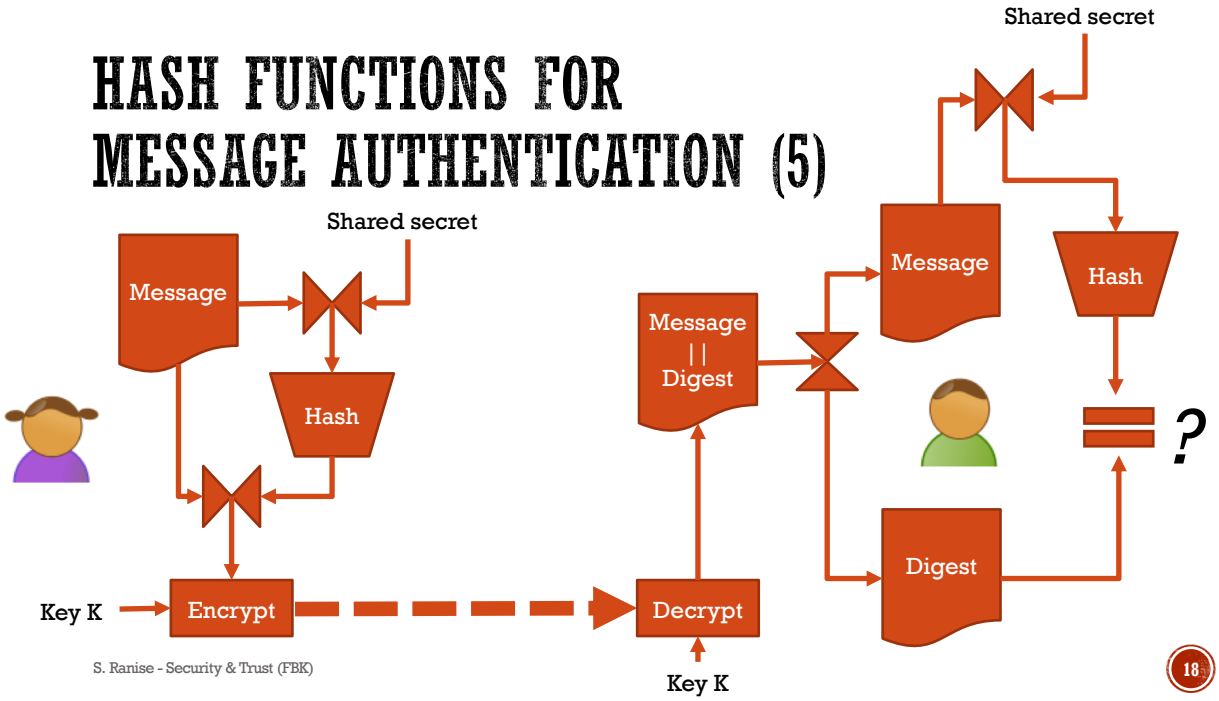
HASH FUNCTIONS FOR MESSAGE AUTHENTICATION (4)



INTERPRETATION OF THE SCHEMA

- **We do not need to assume the availability of neither public nor symmetric key cryptography**
  - In this scheme, no message is encrypted
  - Alice appends a shared secret string known also to Bob, to the message before computing its digest
  - Before checking the digest of the received message for its authentication, Bob appends the same shared secret string to the message
  - It would not be possible for anyone to alter such a message, even when they have access to both the original message and the overall digest
  - Notice that in this case, confidentiality is not considered

HASH FUNCTIONS FOR MESSAGE AUTHENTICATION (5)



INTERPRETATION OF THE SCHEMA

- **We assume the availability of symmetric key cryptography**
  - This scheme is similar to the previous one with the addition of symmetric key cryptography for confidentiality
  - Alice appends a shared secret string known also to Bob, to the message before computing its digest
  - Before checking the digest of the received message for its authentication, Bob appends the same shared secret string to the message
  - It would not be possible for anyone to alter such a message, even when they have access to both the original message and the overall digest
  - Notice that in this case, confidentiality is not considered

AUTHENTICATED ENCRYPTION WITH ADDITIONAL DATA (1)

- When using symmetric ciphers, it is crucial to avoid replay attacks
  - An attacker, even without understanding a ciphertext, can resend it
  - This may have negative consequences
- Example
  - On Sunday, Alice sends the following message to Bob encrypted with a shared symmetric key K
    - You can take holidays tomorrow
  - Bob decrypts the message with K and takes Monday off
  - Eve intercept the encrypted message and resend it to Bob on Monday
  - Bob receives the message on Monday and takes also the following day off
  - On Tuesday, Alice is wondering why Bob is not at work

AUTHENTICATED ENCRYPTION WITH ADDITIONAL DATA (2)

- To avoid the replay attack, what is needed is to **bind the cipher** to a network connection or a session, in order that Eve cannot recreate the same scenario
- With enhanced encryption methods, known as Authenticated Encryption with Associated Data (AEAD), it is possible to both authenticate the cipher and prove its integrity
- For this we provide additional data to authenticate the encryption process, and where we can identify where the ciphertext has been modified, and for it not to be decrypted
- With most conventional AEAD methods we create a nonce value and add additional data (AD) that is authenticated but not encrypted
- Additional data can include addresses, ports, sequence numbers, protocol version numbers, …

AUTHENTICATED ENCRYPTION WITH ADDITIONAL DATA (3)

Encrypted digests are also called Message Authentication Codes (MACs)

- Additional Data allows for binding network packets to the encrypted data, and provide integrity, so that an intruder cannot copy and paste valid ciphertext from other transmissions
- For example, if we bind to a packet sequence number and the port, the authentication would fail for another sequence number or another port
- Examples of AEAD
  - AES-GCM
  - ChaCha20/Poly1305
- Indeed, the wider the range of additional data, the more difficult it will be for an attacker to replicate it

AUTHENTICATED ENCRYPTION WITH ADDITIONAL DATA (4)

- Encryption
  - Input: *plaintext*, *key*, and optionally a *header* in plaintext that will not be encrypted, but will be covered by authenticity protection.
  - Output: *ciphertext* and *authentication tag* (Message Authentication or MAC).
- Decryption
  - Input: *ciphertext*, *key*, *authentication tag*, and optionally a *header* (if used during the encryption).
  - Output: *plaintext*, or an error if the *authentication tag* does not match the supplied *ciphertext* or *header*.
- The *header* part is intended to provide authenticity and integrity protection for networking or storage metadata for which confidentiality is unnecessary, but authenticity is desired

REMARK

- The need for authenticated encryption emerged from the observation that securely combining separate *confidentiality* and *authentication* block cipher operation modes could be error prone and difficult
- This was confirmed by a number of practical attacks introduced into production protocols and applications by incorrect implementation, or lack of authentication (including SSL/TLS)
- There are results that guarantee security (i.e. confidentiality plus authenticity and integrity) of composing encryption and hash functions
  - Those interested can take a look at Chapter 9 of the book

A Graduate Course in Applied Cryptography
by Dan Boneh and Victor Shoup
https://toc.cryptobook.us/

25

THE STRUCTURE OF HASH FUNCTIONS

INTRODUCTION (1)

- All algorithms for computing the digest of a message view it as a sequence of *n*-bit blocks
- The message is processed one block at a time in an iterative fashion in order to generate its digest
- The simplest hash function consists of
  - starting with the first *n*-bit block, perform the bitwise xor with the second *n*-bit block
  - perform the bitwise xor with the next *n*-bit block
  - and so on...
  - This procedure is known as the **xor algorithm** or the **longitudinal parity check** as every bit of the digest represents the parity at that bit position if we look across all of the *n*-bit blocks

INTRODUCTION (2)

- The digest generated by the xor algorithm can be useful as a data integrity check in the presence of completely random transmission errors
- **In the presence of an adversary** trying to deliberately tamper with the message content, the **xor algorithm is useless for message authentication**
- An adversary can modify the main message and add a suitable bit block before the digest so that the final digest remains unchanged
- To see how, $X1, X2, \ldots Xm$ be the bit blocks of a message M, each block of size n bits, i.e. $M = X1 \,||\, X2 \,||\, \ldots \,||\, Xm$

INTRODUCTION (3)

- The digest produced by the xor algorithm can be expressed as

$$D=X1+X2+\ldots+Xm$$

  where + denotes the bit-wise xor operator…
- An attacker can build an arbitrary message $N=Y1||\ldots||Ym-1||Ym$ and then set

$$Ym=Y1+\ldots+Ym-1+D$$

- Indeed, $M||D$ and $N||D$ cannot be differentiated by the recipient of the message
- This is so because the digests of *M* and *N* will be identical as it is easily verified:
  - $X1+\ldots+Xm=D$
  - $Y1+\ldots+Ym-1+(Y1+\ldots+Ym-1+D)=(Y1+Y1)+\ldots(Ym-1+Ym1-1)+D=D$

PROBLEMS DERIVING FROM STRUCTURED MESSAGES (1)

- When hashing regular text and the character encoding is based on ASCII, the collision resistance property of the xor algorithm suffers even more because the **highest bit in every byte will be zero**
- Ideally, one would hope that, with an *n*-bit digest, any particular message would result in a given digest value with a probability of *2^(-n)*
- When the highest bit in each byte for each character is always 0, some of the n bits in the digest will predictably be *0* with the xor algorithm
- This reduces the number of unique digests available and thus increases the probability of collisions

PROBLEMS DERIVING FROM STRUCTURED MESSAGES (2)

- An attempt to avoid this problem is a variation on the basic xor algorithm that consists of performing a one-bit circular shift of the partial digest obtained after each *n*-bit block of the message is processed
  - This algorithm is known as the **rotated-XOR algorithm (rxor)**
- Also this variant suffers from a problem similar to the one described for the original algorithm
- The right way out is to include the length of the message in the digest
  - This can be done by padding the message (we will see how later)

TWO IMPORTANT REMARKS

- Assume the availability of a random message generator
- Assume the availability of a hash function that can be used to compute the digest for each message produced by the generator
- Given a pool of *k* messages produced randomly by the message generator
- Two questions are relevant
  1. what is the value of *k* so that the pool contains at least one message whose digest is equal to a given value *h* with probability *1/2?*
     - *Answer → k=N/2*
     - ***Example**: If N=2^(64), then k=N/2=2^(63), i.e. we must construct a pool of 2^(63) ≈ 9.2*10^18 messages to find in it a digest equals to a given value h with a probability of 50%*
  2. what is the probability that there exist at least two distinct messages in the pool with the same digest*?*
     - *Answer → Probability is 50% when* $k \approx 1.18 * \sqrt{N}$ [Birthday paradox]
     - **Example**: *If N=2^(64), then* $k \approx 1.18 * \sqrt{2^{64}}$ *≈1.18 * 2^(32), i.e. i.e. we must construct a pool of 5.1*10^9 messages to find in it two with equal digests with a probability of 50%*
- For a mathematical justification of the answers, please, see the slides in the digression

BIRTHDAY ATTACK

- Let {*C1, …, Ck*} be the set of variations of *C* [think of C as a contract]
- Let {*F1, …, Fk*} be the set of variations of *F* [think of F as a fraudulent version of C]
- Let *H* be a hash function
- What is the probability that there exists at least one pair *(Ci,Fj)* such that

$$H(Ci) = H(Fj)?$$

- Answer → Probability is 50% when $k \approx 0.83 * \sqrt{N}$, i.e. roughly when k is 2^(N/2)
- If one is willing to generate $k \approx 0.83 * \sqrt{N}$ different versions of both C and F, there will be 50% probability to identify a version of C and a version of F with the same digest… this can give rise to frauds by signing contract C and then replacing it with the fraudulent version of F…

33

DIGRESSION: PROBABILITY OF CERTAIN EVENTS WHEN COMPUTING DIGESTS

QUESTION 1

- Assume the availability of a random message generator
- Assume the availability of a hash function that can be used to compute the digest for each message produced by the generator
- **We are interested in computing the probability that any of the messages is going to have its digest equal to a particular value *h***
- More precisely
  - Given a pool of *k* messages produced randomly by the message generator
  - what is the value of *k* so that the pool contains at least one message whose digest is equal to *h* with probability *1/2?*

ANSWER TO Q1

- To establish the value *k,* we reason as follows
  - Consider that the digest can take on *N* different but equiprobable values
  - Pick a message *x* at random from the pool of messages
  - Since all *N* digests are equiprobable, the probability of a message *x* having its digest equal to *h* is *1/N*
  - Since the digest of message *x* is either equal to *h* or not, the probability of the latter is *1 − (1/N)*
  - If we pick, say, two messages *x* and *y* randomly from the pool, the events that the digest of neither is equal to *h* are probabilistically independent
  - This implies that the probability that none of two messages has its digest equal to *h* is *(1 − (1/N))^2*
  - Generalizing this reasoning to the entire pool of *k* messages, it follows that the probability that none of the messages in a pool of *k* messages has its digest equal to *h* is *(1 − (1/N))^k*

ANSWER TO Q1 (CONT'D)

- Consider the case in which the digests have size of 64 bits
- Then *N=2^(64)* and *k=N/2=2^(63)*
- I.e. we must construct a pool of *2^(63)* messages so to be able to find in it a digest equals to *h* with a probability of *50%*

- Thus, the probability that **at least** one of the *k* messages has its digest equal to *h* is

$$1 - \left(1 - \frac{1}{N}\right)^k$$

- This expression can be simplified to *k/N* by observing that

$(1 + a)^n \approx 1 + a * n$ as *a* gets close to 0, i.e. for increasing values of *N*

- So, given a pool of *k* randomly produced messages, the probability there will exist at least one message in this pool whose digest is equal to a given value *h* is *k/N*
- To answer the original question

what is the value of *k* so that the pool contains at least one message whose digest is equal to *h* with probability *1/2?*

is sufficient to solve the equation *k/N=1/2,* i.e. *k=N/2*

QUESTION 2

- Assume the availability of a random message generator
- Assume the availability of a hash function that can be used to compute the digest for each message produced by the generator
- **We are interested in computing, given a pool of randomly selected messages, the probability that at least two messages in the pool have the same digest**
- More precisely
  - Given a pool of *k* messages produced randomly by the message generator
  - What is the probability that there exist at least two messages in the pool with the same digest*?*

DIGRESSION ON THE BIRTHDAY PARADOX

- Q1: **What is the probability that any of the messages is going to have its digest equal to a particular value *h*?**
  - *What is the probability that, in a class of 20 students, someone else has the same birthday as yours?*
  - Answer: because of the reasoning above, this is roughly equal to 19/365, i.e. around **5.2%**
- Q2: **What is the probability that at least two messages in the pool have the same digest?**
  - *What is the probability that there exists at least one pair of students in a class of 20 students with the same birthday?*
  - Answer: we will see that this is roughly equal to 190/365, i.e. around **52%**
- The answer to Q2 is related to the so-called **birthday paradox**
  - This paradox states that given a group of 20 or more randomly chosen people, the probability that at least two of them will have the same birthday is more than 50%
  - If we randomly choose 60 or more people, this probability is greater than 90%

ANSWER TO Q2

- To answer the question, we reason as follows
  - Let *M1* be the total number of ways in which we can construct a pool of *k* message with no duplicate digests
  - Let *M2* be the total number of ways in which we can construct a pool of k messages allowing for duplicates
  - Then *M1/M2* is the the probability of constructing a pool of *k* messages with no duplicates
  - Subtracting this from 1 yields the probability that the pool of *k* messages will have at least one duplicate digest
  - We are thus left with the problem of computing the values of *M1* and *M2*
  - For M1, we need to find out in how many different ways we can construct a pool of *k* messages so that we are guaranteed to have no duplicate digests in the pool
    - for the first message to insert in the pool, there is a choice of *N* different candidates
    - for the second message to insert in the pool, there is a choice of *N-1* different candidates
    - … and so on

ANSWER TO Q2 (CONTINUED)

- The total number *M1* of ways in which we can construct a pool of k messages with no duplications in digests is *N*(N-1)*…*(N-k+1)* that is equal to
$$\frac{N!}{(N-k)!}$$
- For *M2*, we need to find out the total number of ways in which we can construct a pool of k messages without worrying at all about duplicate digests
  - there are *N* ways to choose the first message to insert in the pool
  - for the second message to insert in the pool, there are still *N* ways
  - … and so on
  - Therefore, the total number of ways we can construct a pool of k messages without worrying about digest duplication is *N*N*…*N* (k times) i.e.
$$N^k$$
- Thus, we have
$$\frac{M1}{M2} = \frac{N!}{(N-k)! * N^k}$$

ANSWER TO Q2 (CONTINUED)

- We are now in the position to say that the probability that the pool of k elements has at least one duplication in the digest is
$$1 - \frac{N!}{(N-k)! * N^k}$$
that can be rewritten as
$$1 - \left(\left(1 - \frac{1}{N}\right) * \left(1 - \frac{2}{N}\right) * \cdots * \left(1 - \frac{k-1}{N}\right)\right)$$
- By remembering that $(1-x) \leq e^{-x}$ for $x \geq 0$, the above expression is **bounded from below** by
$$1 - \left(e^{-\frac{1}{N}} * e^{-\frac{2}{N}} * \cdots * e^{-\frac{k-1}{N}}\right) = 1 - e^{-\frac{k*(k-1)}{2*N}}$$
- When $k \ll N$, i.e. the exponent of e is small, remember that $e^{-x} \approx 1 - x$ and the expression above simplifies to
$$1 - \left(1 - \frac{(k*(k-1))}{2*N}\right) = \frac{k*(k-1)}{2*N}$$

ANSWER TO Q2 (CONTINUED)

- Consider the case in which the digests have size of 64 bits
- Then *N=2^(64)*
- I.e. we must construct a pool of *1.18*2^(32)* messages so to be able to find in it two equal digests with a probability of *50%*

- We are now ready to answer Q2, i.e. to estimate the size k of the pool so that the pool contains at least one pair of messages with equal digests with a probability of 0.5 by solving the following equation:

$$1 - e^{-\frac{k*(k-1)}{2*N}} = \frac{1}{2}$$

that after some simplifications yields $k * (k-1) = 2 * \ln(2) * N$

- When k is large, we have

$$k \approx 1.18 * \sqrt{N}$$

- In conclusion, if the digest can take on a total N different values with equal probability, a pool of $1.18 * \sqrt{N}$ messages will contain at least one pair of messages with the same digest with a 50% probability
  - So if we use an n-bit digest, we have $N = 2^n$ then a pool of *1.18*2^(n/2)* randomly generated messages will contain at least one pair of messages with the same digest with a 50% probability

SCENARIO FOR A BIRTHDAY ATTACK

- Say Mr. Wayne has a dishonest assistant, Mr. Pennyworth, preparing contracts for Mr. Wayne digital signature
- Mr. Pennyworth prepares a legal contract C for a transaction
- Then, Mr. Pennyworth proceeds to create a large number of variations of C without altering the legal content of the contract and computes the digest for each
  - These variations may be constructed by mostly innocuous changes such as the insertion of additional white spaces between some of the words, or contraction of the same; insertion or deletion of some of the punctuation, slight reformatting of the document, etc.
- Next, Mr. Pennyworth prepares a fraudulent version F of the contract C
- As with the correct version, Mr. Pennyworth prepares a large number of variations of F, using the same tactics as before

QUESTION

- **What is the probability that the two sets of contracts will have at least one contract each with the same digest?**
- More precisely
  - Let {*C1, …, Ck*} be the set of variations of *C*
  - Let {*F1, …, Fk*} be the set of variations of *F*
  - Let *H* be a hash function
  - What is the probability that there exists at least one pair *(Ci,Fj)* such that *H(Ci) = H (Fj)?*
- To answer this question, we can reason in a way similar to what we have done above…

SOLUTION (1)

This is the probability that there will **not** exist any digest matches between the two sets of contracts {*C1, C2, . . . , Ck*} and {*F1, F2, . . . , Fk*}

- Assume that all the fraudulent contracts are truly random with respect to the correct versions of the contract
- The probability of *F1*'s digest *H(F1)* being any one of *N* permissible values is *1/N*
- Thus, the probability that the digest *H(C1)* matches *H(F1)* is *1/N*
- The probability that the digest *H(C1)* does not match the digest *H(F1)* is *1 − 1/N*
- Extending the above reasoning to joint events, the probability that *H(C1)* does not match *H(F1)* and *H(F2)* and . . . *H(Fk)* is

$$\left(1-\frac{1}{N}\right)^{k}$$

- The probability that the same holds conjunctively for all members of the set {*C1, C2, . . . , Ck*} is thus

$$\left(1-\frac{1}{N}\right)^{k^2}$$

SOLUTION (2)

- The probability that there will exist at least one match in digests between the set of correct contracts and the set of fraudulent contracts is

$$1 - \left(1 - \frac{1}{N}\right)^{k^2}$$

- Since *1-(1/N)* is always less than $e^{-\frac{1}{N}}$ the above expression will be greater than

$$1 - \left(e^{-\frac{1}{N}}\right)^{k^2}$$

- What is the least value of k so that the above probability is 0.5*?*
- Finding the answer amounts to solving the following equation

$$1 - \left(e^{-\frac{1}{N}}\right)^{k^2} = \frac{1}{2}$$

which gives $k \approx 0.83 * \sqrt{N}$

SOLUTION (3)

- This is called the birthday attack because the combinatorial issues involved are the same as in the birthday paradox
- Note that for an *n*-bit hash coding, the approximate value we obtained for *k* is the same in both cases; i.e. *k = 2^(n/2)*

- So if Mr. Pennyworth is willing to generate $0.83 * \sqrt{N}$ versions of the both the correct contract and the fraudulent contract, there is better than an even chance that he will find a fraudulent version to replace the correct version
- If *n* bits are used for the digest, *N = 2^n* and *k = 0.83*2^(n/2)*

- **The birthday attack consists of Mr. Pennyworth getting Mr. Wayne to digitally sign a correct version of the contract…**
  - This means getting Mr. Wayne to encrypt the digest of the correct version of the contract with his private key
- **… then replacing the contract by its fraudulent version that has the same digest**

48

END OF DIGRESSION: PROBABILITY OF CERTAIN EVENTS WHEN COMPUTING DIGESTS

49

SECURE HASH FUNCTIONS

INTRODUCTION

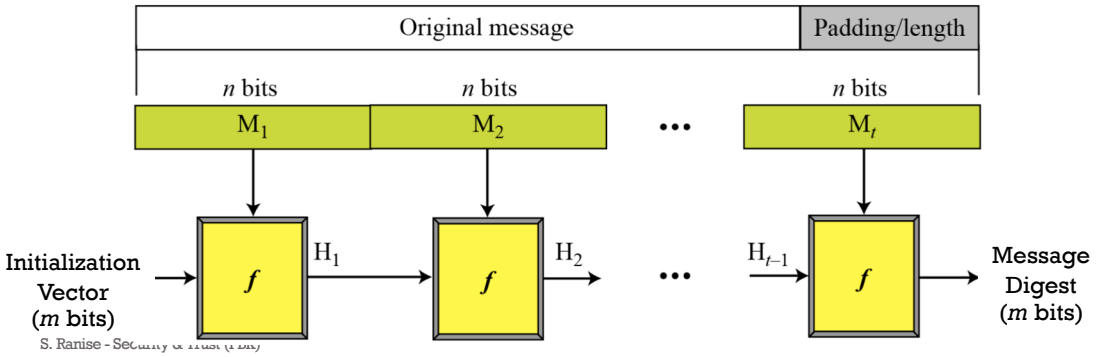
- A hash function is secure if
  - it is computationally infeasible to find collisions
    - I.e. it is computationally infeasible to construct messages whose digest would equal a specified value
  - it is strictly one-way
    - I.e. it lets us compute the digest of a message, but does not let us figure out a message for a given digest (even for very short messages)
- Most secure hash functions are based on the structure proposed by Ralph Merkle in 1979
  - This structure forms the basis of MD5 and the SHA series of hash functions

MERKLE (1)

- The final block also includes the total length of the message whose hash function is to be computed
  - This enhances the security of the hash function since it places an additional constraint on the counterfeit messages

Idea

- The input message is partitioned into t number of bit blocks, each of size n bits
- If necessary, the final block is padded so that it is of the same length as others



MERKLE (2)

- Each stage of the Merkle structure takes two inputs
  - the *n*-bit block of the input message meant for that stage
  - the *m*-bit output of the previous stage
- For the *m*-bit input, the first stage is supplied with a special *m*-bit pattern called the Initialization Vector (IV)
- The function *f* that processes the two inputs, one *n* bits long and the other *m* bits long, to produce an *m* bit output is usually called **compression** function
  - This is so since $n > m$, i.e. the output of the function *f* is shorter than the length of the input message segment
- The compression function *f* may involve multiple rounds of processing of the two inputs to produce the output
- The precise nature of *f* depends on what hash algorithm is being implemented

53

THE SHA FAMILY

SHA = Secure Hash Algorithm

For the next generation SHAs (**not replacing SHA-2 that remains secure and viable**), have a look at https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard

OVERVIEW

This is used in the implementation as we will see in the following

- SHA refers to a family of NIST-approved cryptographic hash functions

		Message size (bits)	Block size (bits)	Word size (bits)	Message digest (size)	Security (bits)
	SHA-1	<2^64	512	32	160	80
SHA-2	SHA-256	<2^64	512	32	256	128
SHA-2	SHA-384	<2^128	1,024	64	384	192
SHA-2	SHA-512	<2^128	1,024	64	512	256

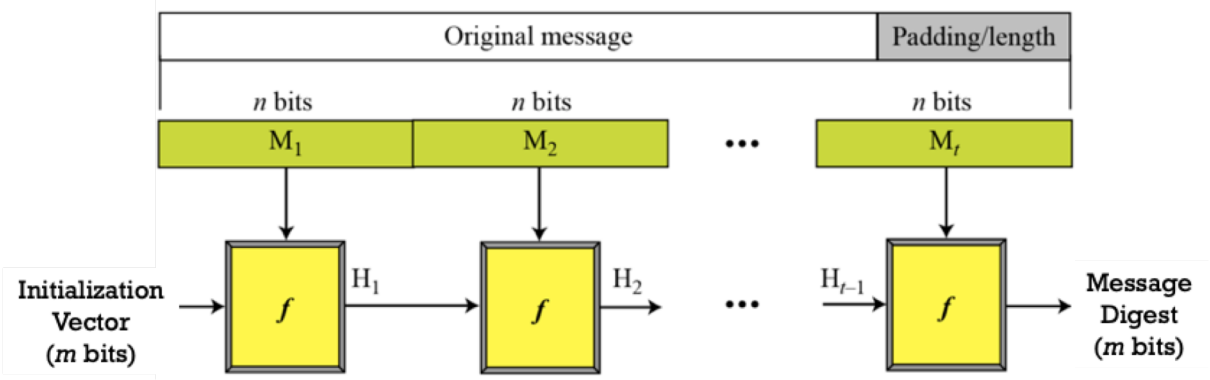
- The **last column** refers to **how many messages would have to be generated before two can be found with the same digest with a probability of 0.5**
- As shown above, for a secure hash algorithm that has no security holes and that produces n-bit digests, one would need to come up with 2^(n/2) messages to discover a collision with a probability of 0.5
  - This is why the entries in the last column are half in size compared to the entries in the Message Digest Size

SHA-1

- SHA-1 is a successor to MD5 that was a widely used hash function
  - There still exist many legacy applications that use MD5 for calculating digests
  - This despite the fact that its usage has been deprecated as it is considered insecure
    - A 2013 attack breaks MD5 collision resistance in 2^18 time
    - This attack runs in **less than a second** on a regular computer
- SHA-1 was cracked **theoretically** in the year 2005 by two different research groups
  - In one of these two demonstrations, it was shown how it is possible to come up with a collision for SHA-1 within a space of size *2^69* only
  - This is much less than the security level of *2^80* associated with this hash function
- In 2017, SHA-1 was broken **in practice**…
  - Full details at https://shattered.io/

SHA-256: STRUCTURE

- *m=512*
- *n=1024*

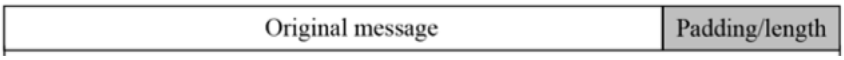


SHA-256: STEPS

- **STEP 1**
  - Pad the message so that its length is an integral multiple of the block size = 1024 bits
  - **Proviso**: the last 128 bits of the last block must contain a value that is the length of the message
- **STEP 2**
- Generate the **message schedule** required for processing a 1024-bit block of the input message
- The message schedule consists of 80 64-bit words:
  - The first 16 of these words are obtained directly from the 1024-bit message block
  - The rest of the words are obtained by applying permutation and mixing operations to some of the previously generated words

SHA-256: STEPS (CONTINUED)

- **STEP 3**
  - Apply round-based processing to each 1024-bit input message block
  - There are 80 rounds to be carried out for each message block
  - For round-based processing
    - store the hash values calculated for the previous message block in temporary 64-bit variables *a, b, c, d, e, f, g, h*
    - in the *i*-th round, permute the values stored in these eight variables and, with two of the variables, mix in the message schedule words and a round constant (see below for details)
- **STEP 4**
  - Update the hash values calculated for the previous message block by adding to it the values in the temporary variables *a, b, c, d, e, f, g, h*



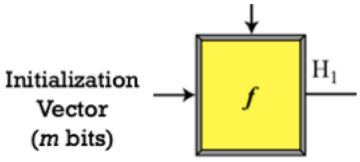
STEP 1: PADDING WITH LENGTH VALUE

- **Goal**: making the input message an exact multiple of 1024 bits
- The last 128 bits of what gets hashed are reserved for the message length value
- Even if the original message were by chance to be an exact multiple of 1024, one still needs to append another 1024-bit block at the end to make room for the 128-bit message length integer
- Leaving aside the trailing 128 bit positions, the padding consists of a single 1-bit followed by the required number of 0-bits
- The length value in the trailing 128 bit positions is an unsigned integer with its most significant byte first
- We denote the padded message by the sequence {*M1, M2, . . . , Mt*}
  - *Mi* is the 1024 bits long *i*-th message block

STEP 2: INITIALIZATION VECTOR

- **Goal**: initialize the hash buffer with IV
- The hash buffer is represented by 8 registers of 64-bits that are labelled by

  a, b, c, d, e, f, g, h



- The registers are initialized by the first 64 bits of the fractional parts of the square-roots of the first eight primes (see gray box on the right for their hexadecimal representation)

- 6a09e667f3bcc908
- bb67ae8584caa73b
- 3c6ef372fe94f82b
- a54ff53a5f1d36f1
- 510e527fade682d1
- 9b05688c2b3e6c1f
- 1f83d9abfb41bd6b
- 5be0cd19137e2179

STEP 3: PROCESS A 1024-BIT MESSAGE BLOCK (1)

Bitwise xor

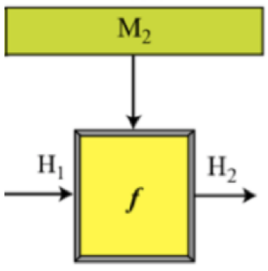
Addition modulo *2^64*

- **Goal**: iteratively perform round processing 80 times
- Implemented in the function *f*
- **Preliminarily, generate the message schedule**
  - I.e. generate 80 words *W0, …, W79* of 64 bits size each
  - *W0, …, W15* are the sixteen 64-bit words in the 1024-bit message block *Mi*
  - *W16, …, W79* are obtained by

$$W_i = W_{i-16} +_{64} \sigma_0(W_{i-15}) +_{64} W_{i-7} +_{64} \sigma_1(W_{i-2})$$

where

$$\sigma_0(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x)$$
$$\sigma_1(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x)$$



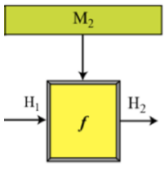
- $ROTR^k$: circular right shift of the 64 bit arg by *n* bits
- SHR^k: right shift of the 64 bit arg by n bits with padding by 0 on the left

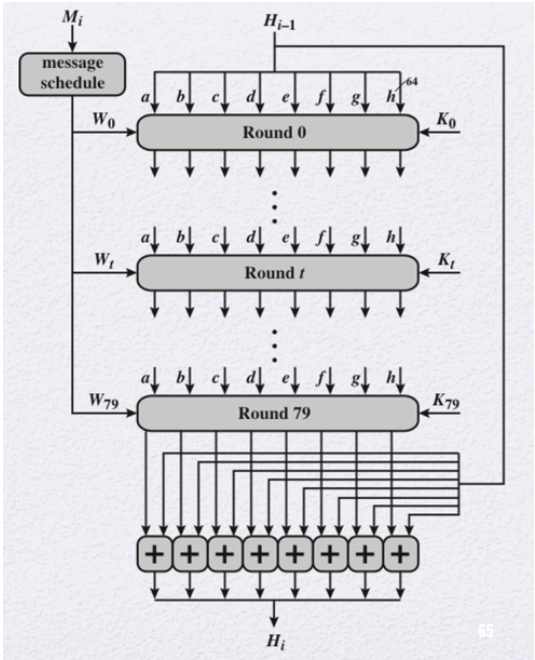
STEP 3: PROCESS A 1024-BIT MESSAGE BLOCK (2)

- In the round-based processing of each input message block, the *i*-th round is fed the 64-bit message schedule word *Wi* and a special constant *Ki*
- The constants *Ki*'s represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers
  - These constants are meant to be random bit patterns to break up any regularities in the message blocks:

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcbd41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210	b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725	06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926	4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8	81c2c92e47edaee6	92722c851482353b
a2bfe8a14cf10364	a81a664bbc423001	c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910	f40e35855771202a	106aa07032bbd1b8
19a4c116b8d2d0c8	1e376c085141ab53	2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb	5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60	84c87814a1f0ab72	8cc702081a6439ec
90befffa23631e28	a4506cebde82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273eceea26619c	d186b8c721c0c207	eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a	5fcb6fab3ad6faec	6c44198c4a475817

STEP 3: PROCESS A 1024-BIT MESSAGE BLOCK (3)



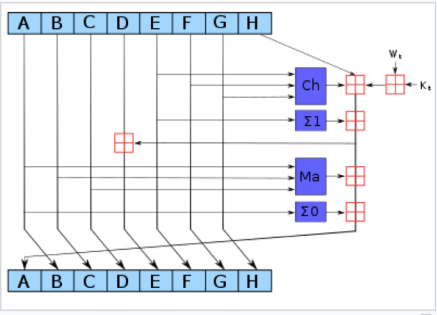


- The module *f* (depicted on the right) has two inputs:
  - the current contents of the 512-bit hash buffer *Hi-1*
  - the 1024-bit message block Mi
- The module *f* consists of 80 rounds of processing: how the contents of the hash buffer are processed along with the inputs *Wi* and *Ki* is referred to as the **round function** *0, 1, …, 79*
- The round function consists of a sequence of **transpositions** and **substitutions** designed to **diffuse** to the maximum extent possible the content of the input message block

ROUND FUNCTION DEFINITION

- The details of the round function are depicted on the right
  - The register variables *a, …, h* are shown in capitals
- For a complete description, see

https://csrc.nist.gov/publications/detail/fips/180/4/final



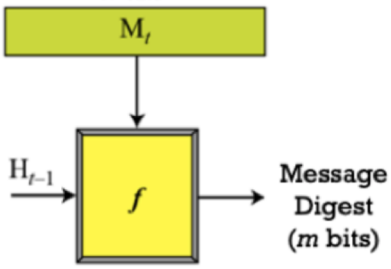
One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$
$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$
$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$
$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256.
The red ⊞ is addition modulo 2^{32} for SHA-256, or 2^{64} for SHA-512.

STEP 4: OUTPUT

- After all the *t* message blocks have been processed, the content of the hash buffer is the message digest



To play with SHA-512 and study a Javascript implementation of SHA-512, see
https://www.movable-type.co.uk/scripts/sha512.html

68

FINAL REMARKS ON HASH FUNCTIONS

FINAL REMARKS ON HASH FUNCTIONS (1)

- In reality, hash functions are not close to one-way functions for which
  - it is is easy to compute the digest
  - but it is computationally hard to compute a **preimage**:
    - given a digest *h(a)*, find *b* such that *h(b)=h(a)*

- **There is no proof that one-way hash functions exist, or even concrete evidence that they can be constructed**
- Pragmatically, there are example hash functions (e.g., SHA-2 and SHA-3) that seem to be one-way hash functions that are easy to compute but we know of no easy way to find a preimage

FINAL REMARKS ON HASH FUNCTIONS (2)

- Similar situation for the other properties of hash functions, namely
- **second pre-image computation**, i.e.
  - it is hard to find *b* given *a* such that *h(b)=h(a)*
- **collision resistance**, i.e.
  - it is hard to find *a* and *b* such that *h(a)=h(b)*

- There is no proof that hash functions satisfying these properties exist, or even concrete evidence that they can be constructed
- Pragmatically, there are examples that seem satisfy them: they are easy to compute but we know of no easy way to find a second preimage or to break collision resistance

FINAL REMARKS ON HASH FUNCTIONS (3)

- Despite all these issues, for evaluating the theoretical security of hash functions, the random oracle model is used
- A random Oracle is a function that gives fixed length output
  - If it is the first time that it receives a particular input, it gives random output
  - If it already has received the input, it will repeat the corresponding output
  - This is the ideal strongly collision resistant function
- It can be used to prove security of encryption/signing under the assumption that the functions used behave like a random oracle
- This is usually treated as strong evidence rather than a real proof
- The evidence falls if weaknesses are found in the functions used

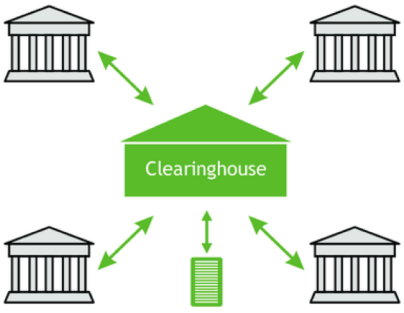
72

APPLICATION TO CRYPTOCURRENCIES

Cryptocurrency

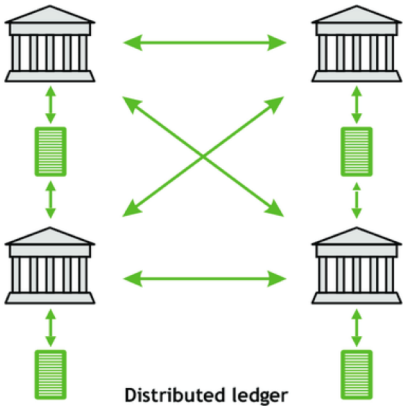
Cryptocurrency is a digital currency in which encryption techniques are used to regulate the generation of units of currency and verify the transfer of funds, operating independently of a central bank.

INTRODUCTION: A PARADIGM SHIFT



Centralised ledger

- Central authority has ultimate control over data
- **Central authority shall be trusted**



Distributed ledger

- Pre-defined set of rules has control over data
- **Implementation of consensus algorithm shall be trusted**

IN THE BEGINNING…

- Satoshi Nakamoto, "*Bitcoin: A Peer-to-Peer Electronic Cash System*," November **2008**. Available at https://bitcoin.org/bitcoin.pdf

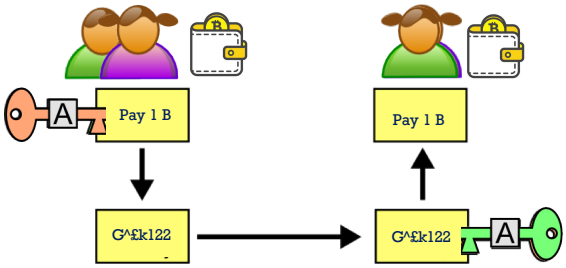
- First release of bitcoin's code, establish the network, disperse the first bitcoins, and unveil the world's first blockchain

- Few weeks before, Lehman Brothers collapsed

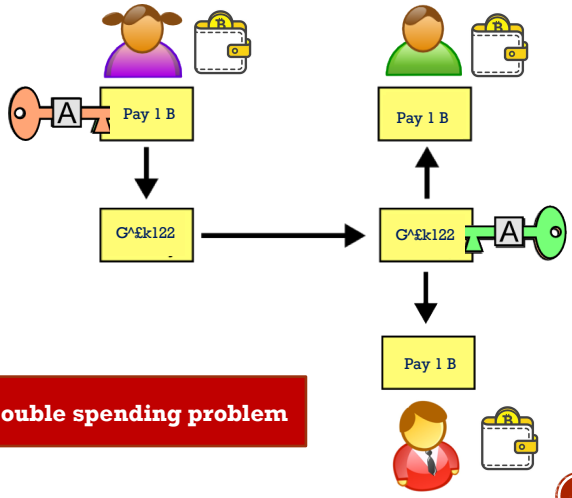
PKC AND BLOCKCHAIN: INITIALIZATION

- Alice has a wallet with pair of **private** & **public** keys
- Public key is her public address

PKC AND BLOCKCHAIN: A TRANSACTION



- Alice encrypts transaction with her **private** key and send the result to Bob
- Bob decrypts the message with Alice **public** key
- **Bob (as anyone else on the network) can use Alice public key only to confirm that the transaction request came from Alice**



Double spending problem

A SOLUTION TO DOUBLE SPENDING

- **Mining**
- **Proof of work**: solving **mathematical puzzles** based on the notion of (cryptographic) hash function
- **Example problem**: find the number appended to the input "*Diamonds are a girl's best friends*" that gives a digest starting with a sequence of 4 zeros
- How to solve this?
- *Since the hash function makes it impossible to predict what the output will be*, one needs to **guess**…

GUESSING

1. INPUT: "Diamonds are a girl's best friends1"
   OUTPUT: "054b9989acff07761ddb2df8a0565aa4" ✓
2. INPUT: "Diamonds are a girl's best friends2"
   OUTPUT: "d162e4e0a9d25111812ee0bed5b1936e" ✓

…

✓

GUESSING

1. …iends1"
   …a0565aa4"
2. …nds2"
   …ed5b1936e"

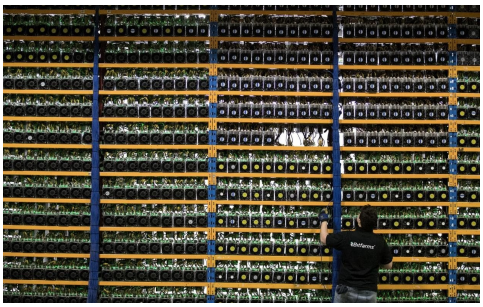
…

The **electric power consumed** during the computation required to solve the crypto-puzzle gives **value** to the **bitcoin** transaction

✓

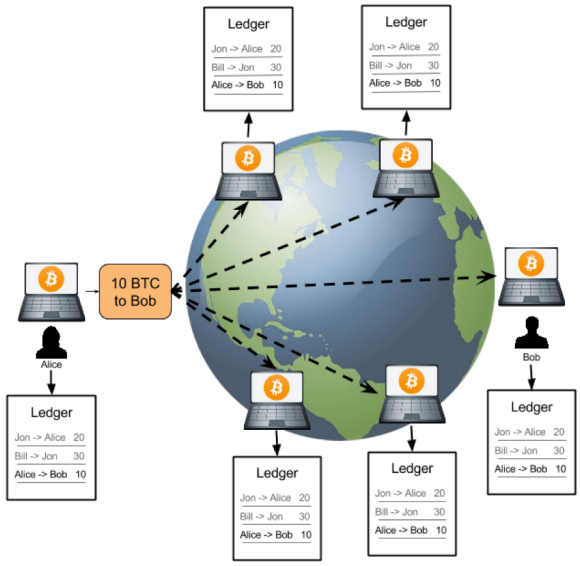
INCENTIVE FOR MINERS

Miners receive bitcoins for solving crypto-puzzles



ADDING THE TRANSACTION TO THE LEDGER

- Bitcoin blockchain can be thought of as a decentralized database that is managed by distributed computers on a peer-to-peer network
  - **Each peer maintains a copy of the ledger**: updates and validations are reflected in all copies simultaneously

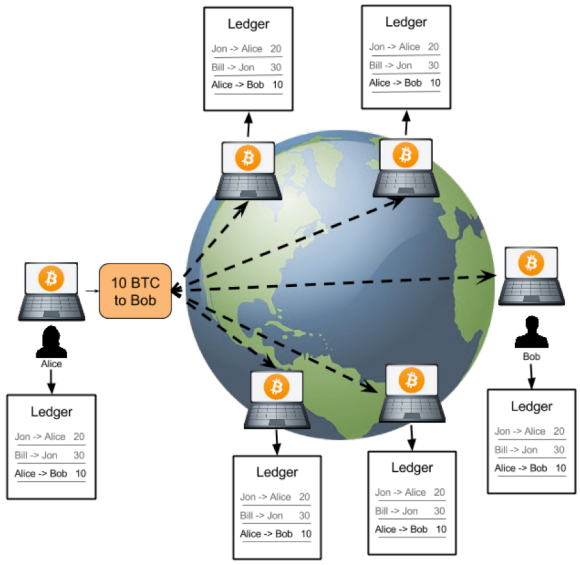


Picture from https://www.ybrikman.com/writing/2014/04/24/bitcoin-by-analogy/

ADDING THE TRANSACTION TO THE LEDGER

What if someone is cheating or misunderstood the message?

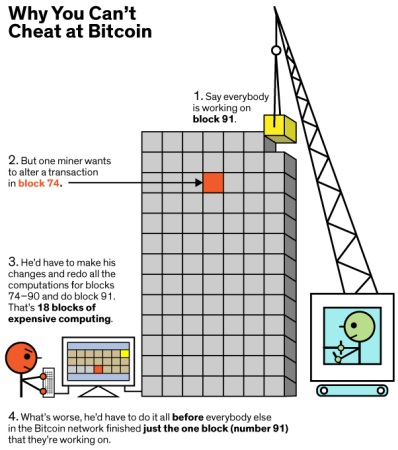
Coherence in the decentralized ledger is maintained by a **consensus algorithm**!



CONSENSUS ALGORITHMS

- A consensus algorithm is a process used to achieve agreement on a single data value among distributed processes or systems
- Consensus algorithms are designed to achieve reliability in a network involving multiple unreliable nodes

CONSENSUS FOR THE BLOCKCHAIN

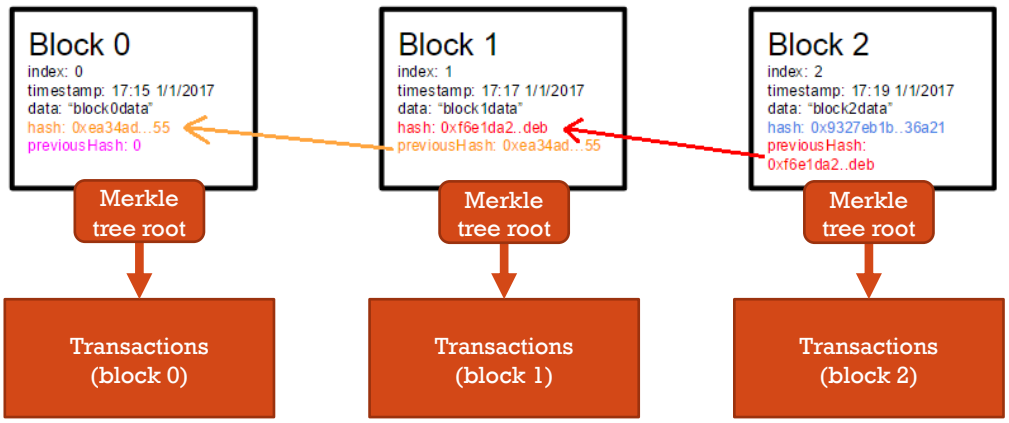


- Bitcoin uses the proof of work algorithm to ensure security in a trustless network, by including mechanisms that guarantee that the effort of mining is represented within the block submitted by the miner
  - Although any party can submit a chain of blocks to the ledger, **the amount of computing resources required to fake consensus is too great to make it worthwhile to a dishonest party**

WHAT IS A BLOCKCHAIN?

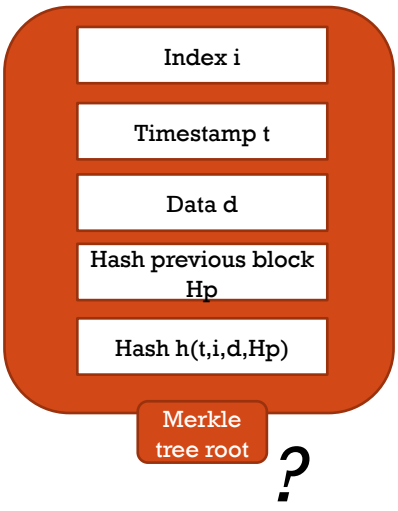
- It is a **decentralized, distributed and public digital ledger** that is used to record transactions across many computers so that **records cannot be altered retroactively** without the alteration of all subsequent blocks and the **consensus** of the network
- in other words, it is a **distributed database** that maintains a continuously **growing list of ordered records**, called **blocks**
  - Blocks are linked using cryptography
  - Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data
- Other applications beyond cryptocurrencies
  - monitoring of supply chains
  - healthcare
  - digital identities (wallets as in eIDAS 2.0)
  - …

BLOCKCHAIN DATA STRUCTURE…



ANATOMY OF A BLOCK (AGAIN)

Furthermore…
Anyone modifying a datum in a block will invalidate the chain in which the block occurs as the hash of each block is computed by taking into consideration the hash of the previous block



Why including the hash of a previous block?

Integrity: if someone modifies datum d, the block is invalidated as the hash will not match! **Furthermore…**

Recall that
- Small variations in input generate widely different outputs

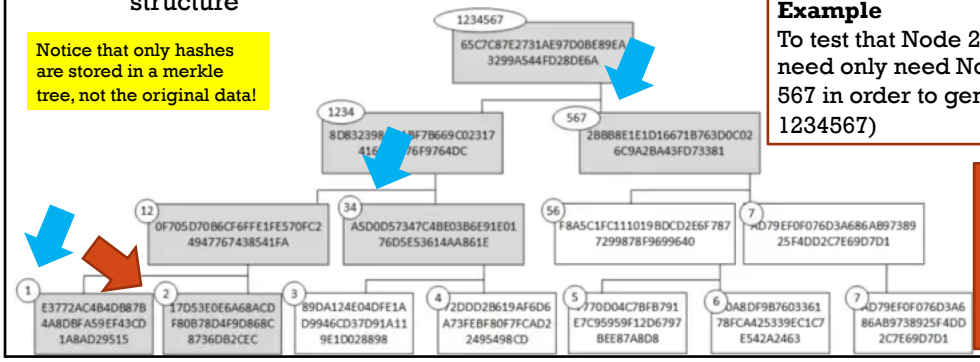
WHY MERKLE TREES?

- Assume a transaction is associated with just one transaction…
  - Then, every node on the network would have to retain a complete copy of every single transaction ever made
    - Indeed, this is quite an amount of information
  - To confirm that there were no modifications, a computer used for validation would need a lot of computing power to compare ledgers
- To avoid this kind of problems, a bunch of transactions (taking place more or less at the same time) is associated to a block…
- … but then you need an efficient data structure to verify that a single transaction is indeed in that bunch of transactions…
- … such data structure could be a Merkle tree!

HOW DOES A MERKLE TREE WORK

- **Definition**: a tree in which
  - every "leaf" (node) is labelled with the cryptographic hash of a data block
  - every node that is not a leaf (called a branch, inner node, or inode) is labelled with the cryptographic hash of the labels of its child nodes
- A hash tree allows efficient O(log n) and secure verification of the contents of a large data structure

Notice that only hashes are stored in a merkle tree, not the original data!



Example
To test that Node 2 is in the tree, one would need only need Node 1, Node 34 and Node 567 in order to generate the root (Node 1234567)

A Merkle tree is an efficient example of a **cryptographic commitment scheme**, in which the root of the tree is seen as a commitment and leaf nodes may be revealed and proven to be part of the original commitment

99

HASHING FOR CRYPTOCURRENCIES

PROOF OF WORK (POW)

- Hashing finds extensive use in crypto currency algorithms
  - **Proof-of-work** calculations for **mining** a new coin
  - Authenticate ownership of coins
  - Establish a protocol for transferring the ownership from one node to another in a crypto currency network
  - Create a protection against the **double spending** problem
    - No node should be able to sell the same coin to multiple other nodes in the network

- **Proof of Work (PoW)**
  - algorithm that generates a result by **consuming a lot of computational resources**
  - the validity of the result is cheap to verify
- Usually, PoW is based on solving mathematical problems requiring an exhaustive search in the space of all possible solutions

Example: find a number that concatenated to a string gives a hash beginning with a sequence of 4 zeros

FEATURES OF PROOF-OF-WORK

- **Probabilistically**, the actor (called **miner**) who will solve the puzzle first the majority of the time is the one who has access to the most computing power
- While it is hard to find a solution for the puzzle, it is easy to verify that the solution is correct
- **A miner gets rewarded with some currency (block reward, transaction fees) and thus are incentivized to keep mining**
- The other nodes verify the validity of the block by checking that the hash of the data of the block is less than a given number
- It is very energy consuming*!!!*

AUTHENTICATION OF OWNERSHIP

- Authenticate ownership of coins
  - Consider a transaction involving a seller and a buyer (two nodes of a crypto coin network) among which a coin should be exchanged
  - A node in a crypto coin network may establish ownership of a coin by first taking a hash of all the relevant parameters related to the coin and encrypting the hash with its private key
  - Transferring the ownership of a coin from one node to another in a coin network may be established by the receiving node sending its public key to the selling node
  - The selling node may then append that public key to the coin, take a hash of the resulting structure, encrypt the hash with its private key, and send the resulting object to the buyer
- Hashing also plays a critical role in preventing double spending
  - More on this later

CRYPTO COINS

- The coins of a virtual currency are **rare** software objects created by PoW calculations
- Digital signatures based on Public key cryptography are crucial for the validation of transactions
- The notion of a transaction comes with a verifiable proof of ownership transfer from the seller of a coin to its buyer
- The notion of a block as a set of transactions and, even more importantly, the notion a blockchain that is used to implicitly embed in each block a hash of all the previous transactions carried out so far
  - This provides security against the double spending problem and makes it possible to use crypto currencies in a manner similar to real currencies

ON RARITY (1)

- Is it possible to create rare digital objects*?*
- The requirements for a rare software object are the following
  - it must be extremely difficult to produce the object
  - it must be extremely easy to verify that it was extremely difficult to produce the object
- Interestingly, we can use hash functions to satisfy both these computational requirements
  - If we require the rare digital object to possess a hash value that has a particular pattern to it, one shall mount a large computational effort to discover a message whose hash would correspond to that pattern
  - However, after finding such a message, it would be trivial for anyone to verify the discovery

ON RARITY (2)

Notice that requiring a hash value to be less than a certain integer is equivalent to requiring that a certain number of the most significant bits of the digest are 0s

- Consider an *N*-bit hash function
- Consider the following criterium for rarity

  the digest be smaller than some integer t
- An *N*-bit digest can be seen as an integer between 0 and *2^N-1*
- The results of a good hash function should be similar to a random generator that uniformly distributes the outputs in this range
- Then, the probability that one discovers a message whose digest is upper bounded by *t* is given by *t/2^N*
- This probability can be made as small as desired by making *t* sufficiently small for any reasonable value for *N*
  - Example: for *t=1000* and *N=256*, the probability is pretty close to 0

OWNERSHIP OF COINS

- How can we establish the ownership of coins (rare digital objects)*?*
- Preliminarily, we need to figure out how to establish **digital identities for the human owners** of the rare objects in the digital world
- A human user's digital identity can be the hash of his/her public key
- Since the public key will be available to all who wish to verify that identity
  - For someone to verify an id, all they have to do is to hash the public key of the individual and see if it yields the same value as the id under examination
- For ownership, all the coin discoverer has to do is to compute the digest of the coin, encrypt it with his/her private key, and append the signature to the coin
  - Does this look problematic?

TRANSFER OF COINS (1)

- The ownership of a coin from a seller to a buyer can be established
  - by the seller asking the buyer for his/her public key,
  - incorporating that key as an additional field in the coin,
  - then hashing the thus augmented coin,
  - finally, with the seller encrypting the augmented coin with his private key

- This procedure was suggested by Satoshi Nakamoto for Bitcoin transactions
  - The original paper by Satoshi Nakamoto can be downloaded at
    https://bitcoin.org/bitcoin.pdf

TRANSFER OF COINS (2)

The meaning of this sentence will be clarified in the following slides

- After a node has accumulated a certain designated number of outgoing transactions, it packages them into a block
- In contrast with a transaction that goes to one specific other node in the network, a block is broadcast to the entire network
- Observations about block management
  - One of the most important properties of a block is the blockchain length that is associated with it
  - For the very first block that is broadcast to the network, the length of the blockchain is equal to the number of transactions in the block
  - **After a block is accepted by the network**, the nodes in the network have no choice but to use that block as a genesis for discovering the next coin
  - Should a node decide to ignore the accepted block and continue using a previously accepted block in its coin mining algorithm (with a shorter blockchain), any transactions and blocks created by the code would have associated with them shorter blockchain length
  - These would eventually be rejected by the network in favour of transactions and blocks with longer blockchains

TRANSFER OF COINS (3)

- What is the meaning of the sentence "after a block is accepted by the network"*?*
- The meaning is that it should be possible for any node in the network to verify that a block is authentic
- The authenticity verification process turns out to be recursive since
  - every new block has embedded in it a signed hash of the previous block whose hash was used as a genesis for the new coin mined that are in the transactions in the new block
  - every block has a unique identifier (BlockID) associated with it that makes it simple to verify that the new block is a valid construction from the previous block
  - this process can be continued recursively backwards until the the first block that was accepted by the network (called the genesis block) is reached

TRANSFER OF COINS (4)

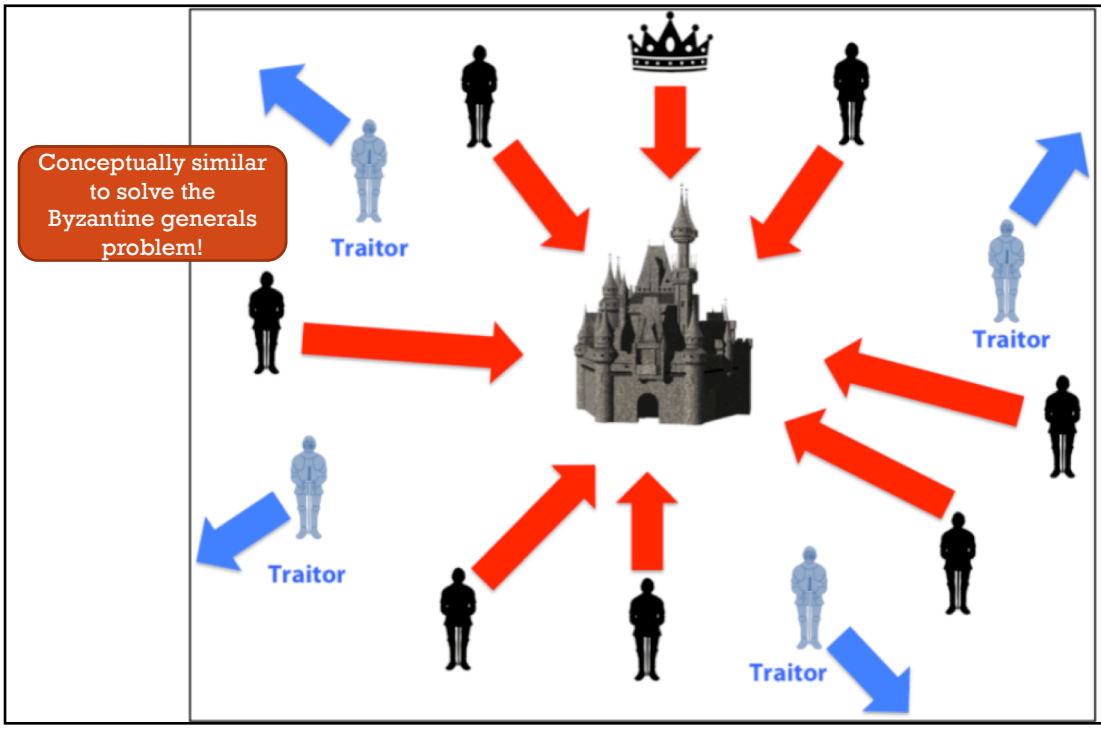
- After a node has accumulated a designated number of transactions, it packages them into a block that is then broadcast to the entire network
- As to how many transactions to pack into a block is a decision that each node must make for itself
- The advantage for choosing a relatively large number for the number of transactions that go into a new block is that its acceptance by the network would bring to an end abruptly a large number of ongoing mining runs at the other nodes
- However, there is also a risk associated with waiting too long for packaging the transactions into a new block
  - The node may receive a new block from some other node that increases the length of the blockchain that it is currently using in the coin mining algorithm and it will need to suddenly abandon its ongoing mining search and start a fresh one with a new genesis

TRANSFER OF COINS (5)

- The search for a new coin is a completely random process in which the time taken to discover the next coin cannot be predicted in advance
- The ongoing search for a new coin must be abandoned immediately when a block of transactions is received from the network if the received block has a blockchain that is longer than what is being used in the current search, or, for a given blockchain length, if the received block has a PoW difficultly that is greater than what is being used in the current search
- These imply that, at the least, each node must execute
  - the coin mining code
  - the code that is in charge of receiving and validating incoming blocks

  in two separate threads or processes
- Since a node can construct a new block using a mixture of transactions, some of which are based on the coins discovered by the client while others are based on the coins acquired from other clients, the node is going to have to also run the block construction code in a separate thread or a process of its own

FAULT TOLERANT CONSENSUS



BYZANTINE GENERALS PROBLEM

- A group of generals must decide whether to attack or retreat
- Some generals may prefer to attack, while others prefer to retreat
- The important thing is that every general agrees on a common decision, for a halfhearted attack by a few generals would be much worse than either a coordinated attack or a coordinated retreat
- The problem is complicated by the presence of **treacherous generals** who may not only cast a vote for a suboptimal strategy, they may do so selectively
  - Example: if 9 generals are voting, 4 of whom support attacking while 4 others are in favor of retreat, the 9th general may send a vote of retreat to those generals in favor of retreat, and a vote of attack to the rest…
- The problem is complicated further by the generals being physically separated and having to send their votes via **messengers who may fail to deliver votes or may forge false votes**

http://lamport.azurewebsites.net/pubs/byz.pdf

A SIMPLIFIED VERSION OF THE PROBLEM

- **Two generals problem**
  - General 1 is the leader
  - General 2 is the follower
  - Each general's army on its own is not enough to defeat the enemy army successfully, thus they need to cooperate and attack at the same time
- In order for the two generals to communicate and decide
  - General 1 has to send a messenger across the enemy's camp that will deliver the time of the attack to General 2
  - There is a possibility that the messenger will get captured by the enemies and thus the message will not be delivered resulting in General 1 attacking while General 2 not
- Even if the first message goes through, General 2 has to *acknowledge* that he received the message, so he sends a messenger back, thus repeating the scenario where the messenger can get caught
- This extends to infinite chains of acknowledgements with generals unable to reach agreement

BYZANTINE FAULT TOLERANCE

- Byzantine Fault Tolerance is the characteristic which defines a system that tolerates the class of failures that belong to the Byzantine Generals' Problem
- Byzantine Failure is the most difficult class of failure modes as it implies no restrictions, and makes no assumptions about the kind of behavior a node can have
  - Example: a node can generate any kind of arbitrary data while posing as an honest actor
- Byzantine Fault Tolerance is needed in airplane engine systems, nuclear power plants, …
  - Any system whose actions depend on the results of a large amount of sensors
- There exists algorithm that solves the problem under the assumption that only 1/3 of the nodes involved can be failing

RELATIONSHIP WITH BLOCKCHAIN

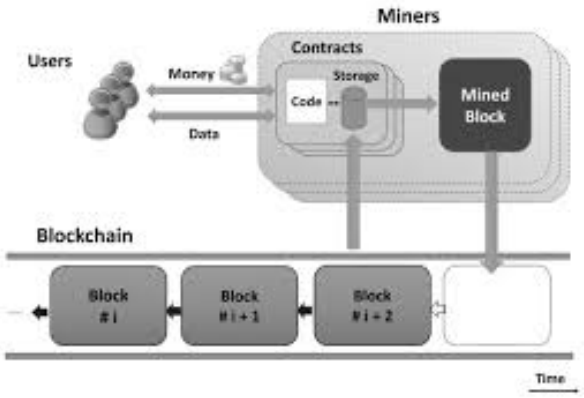
- Blockchains are decentralized ledgers without a central authority
- Attackers have huge economic incentives to try and cause faults
- Thus Byzantine Fault Tolerance is a basic requirement for blockchains
  - Without, a peer is able to transmit and post false transactions effectively nullifying the blockchain's reliability
  - Since there is no central authority to take over and repair the damage, the consistency of the transactions stored in the blockchain becomes void
- Bitcoin is Byzantine Fault Tolerant by solving the Byzantine Generals Problem by using the **Proof-of-Work** approach

118

DIGRESSION ON TRUSTING TRUST

A BC/DLT is a piece of SW…

… as well as the smart contracts running on top of it!



THIS HAS SECURITY IMPLICATIONS!

BC/DLT are SW

- SW contains bugs that may lead to vulnerabilities that can be exploited by attackers

August 2010: Bitcoin Hack

Bitcoin investors are well aware that the cap on Bitcoin supply is 21 million. In 2010, one hacker was able to exploit a bug in Bitcoin's software and create a single block involving a transaction of 184 billion Bitcoin. The bug was patched out shortly thereafter.

- **Smart-contracts** are also SW

Last November, a user accidentally froze around 514,000 ETH (worth approx. $155 million) in Parity, a popular Ethereum wallet. The culprit was a bug in the wallet's software, which the user in question accidentally triggered.

This wasn't the first incident on the Parity wallet. Only months earlier, another bug in Parity's software enabled hackers to get away with 150,000 ETH (approx. $30 million). Neither is Parity the only Ethereum application that has suffered from smart contract vulnerabilities.

In 2016, in the famous DAO hack, hackers got away with 3.6 million ETH, 15 percent of all Ether in circulation at the time. The incident caused a rift in the Ethereum community and resulted in a hard fork, creating two versions of the famous cryptocurrency.

122

END OF DIGRESSION ON TRUSTING TRUST