14.129
Spring, 2025
Lecture 9

Modern Encryption: Key Concepts

Public and Private Keys, Encoded Messages, Hashes, Cryptographic Puzzles, Fully Homomorphic Encryption, Multiparty Computation, Zero Knowledge Proofs

Robert M. Townsend

Elizabeth & James Killian Professor of Economics, MIT

Massachusetts Institute of Technology

Outline

- Encryption is a tool in and of itself, separate from ledgers and so on
- Encoded message systems
- Hashes
- Cryptographic puzzles
- Modern encryption, public private keys, cyclic rings
- Fully homomorphic encryption
- Multi-party computation
- Zero knowledge proofs
- Encryption as utilized in Bitcoin
  - Merkle Trees, Block ID hashes in proof of work
- Summary of FHE, MPC, ZKP and other strategies

Encryption: Programmed Contracts as Distinct from Distributed Ledgers

- Encryption is the third leg of distributed ledgers, in the validation protocols and data generated – validation on ledgers
  - Encryption allows the ledgers as financial accounts to be altered with e-transfers of money and assets in a secure way, without assuming trust, creating immutable and readily indexed records.
- Encryption is a key part of programmed contracts, dealing with private information, limited communication, and limited commitment and with confidential data that need to be secure and are costly to storage and retrieve - contracts in environments with obstacles to trade
  - Encryption is a way to implement optimized solutions to bilateral and multi-agent mechanism design problems where message need to be kept secret, data needs to be secured and immutable, and where agents can commit to arrangements and commit to the way they are implemented, without inconsistencies, even if they would have liked to renege.

Encryption Solves the Secure Communication Problem

Incoming and outgoing messages rely simply on the distinction between public and private keys.

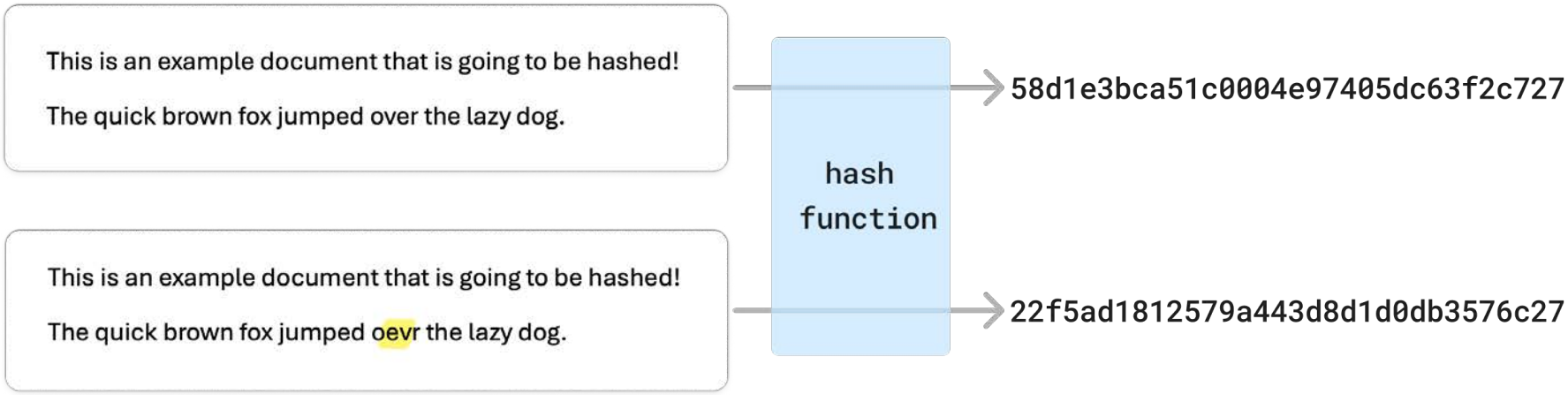
First: Encryption of Messages

- A public key in the context of a specified mathematical space is used to encrypt messages into a ciphertext.
- The ciphertext are public.
- A private key in that space is used for decryption of messages.
- Algorithms on the designated space make it virtually impossible to decipher an encrypted message without the private key that was used as part of generating it.
  - The analogy is that the ciphertext, like product of two primes, is the encrypted message, virtually meaningless on its own.
  - The private key is like one of the factors.

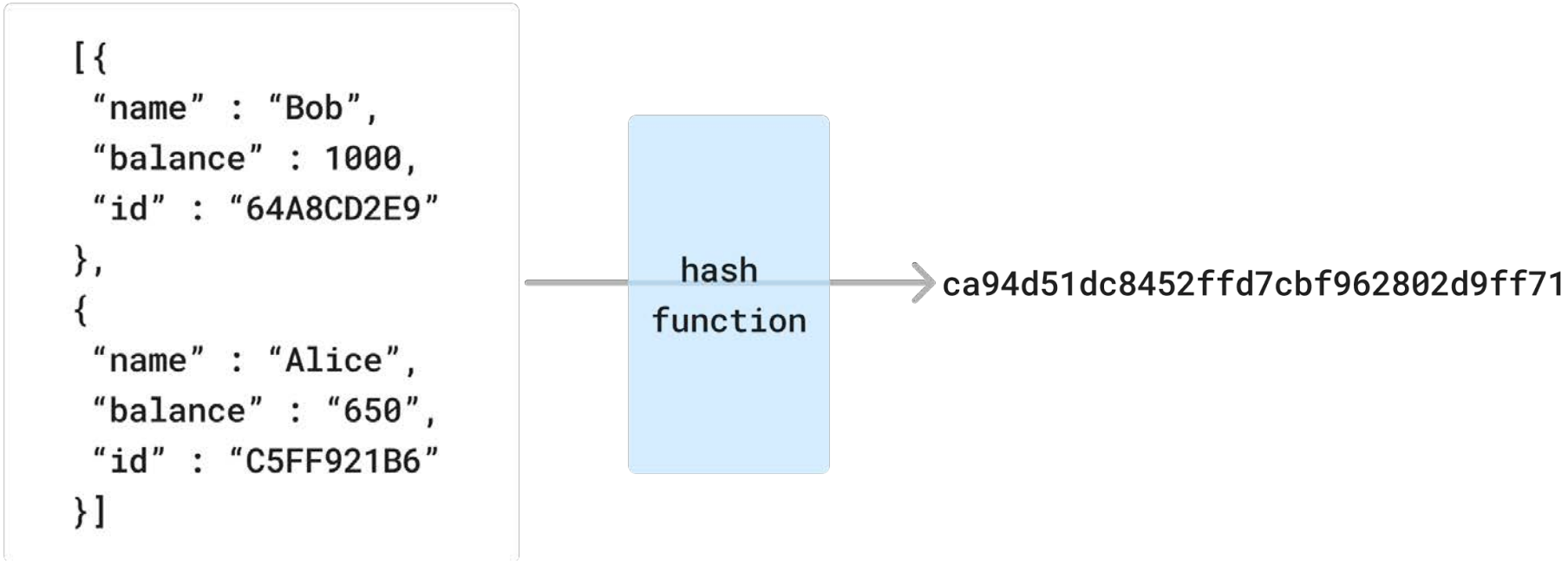
Cryptographic Hash Functions

- A cryptographic hash function is an algorithm that takes an arbitrary amount of input and produces a fixed-size output, a hash.
  - A given input always generate the same hash value or enciphered text.
  - A hash cannot be deciphered, except by random guessing;
- A good hash should:
  - make it very hard to reconstruct the original input from the output or hash (Non-reversibility, or One-way Function).
    - With a change in just one bit of the original input, the output of enciphered text should change significantly and unpredictably.
    - Thus any tinkering is detected (Diffusion, or Avalanche Effect).
  - be hard to find two different inputs that hash to the same enciphered text (Collision Resistance).
  - should not be predictable from the input, for security of the data (Nonpredictable).

One Way Function



Hashing Data



Hashes as a Signature System

- Hashes can be used as a kind of signature system, not only for messages, as above in the outgoing message signature scheme, but also for documents or data sets.
  - Suppose one has an original data set that is stored on some server. The hashed version can be public as it does not reveal the data.
  - A person with permission to access the underlying data wants to be sure when accessing the data, as a candidate file, that the data files have not been corrupted nor deliberately altered.
  - If the original data are signed by their unique hash, then the candidate file can be hashed with the same known function as the original and the two hashes compared.
    - Think of comparing original data with candidate versions, but here it is done with the hashes.

Hashing is a Way of Securing Documents, Data Sets, and Contracts with a Seal

- Hashing
  - thinking of the two prime numbers when multiplied together as encrypted ciphertext, a unique hash, which is impossible to factor.
- The hash can serve **as evidence, a seal**
  - e.g., as an immutable seal on a document. Someone claims to be the creator of the document, gives a third, external party one of the two factors as evidence. The second factor is now easily deduced, and indeed when multiplied give the original hash, the two factors is easily verified. Thus, the creator of the document makes a commitment to authorship.
- Indeed, the hash serves **as a certification** as any outsider can ask to see one of the factors, but does not actually need to do so. As with signature schemes, the hash allows validation of the author.
- The **entire document can be hashed**.
  - With the hash made public, hash serves as validation that the document itself has not been altered, as one can always look and compare the original hash against the hash of a potentially worrisome forgery. This also solves what otherwise would be a limited commitment problem, that the creator might like to misrepresent an agreement or change data.

Third: Cryptographic Puzzles

- Cryptographic puzzles: if the problem is hard, but not too hard, the degree of which can be controlled, then the problem can be solved by trial and error.
  - The prime numbers in our running example could be large but not too large. Then trial and error takes time and energy, requiring enormous computational power. Thus a problem can be parameterized to control its difficulty and to make who succeeds in the trial and error procedure essentially random.
- Cryptographic puzzles are one of the better known parts in Bitcoin's validation protocols, the mining and associated utilization of energy.

Factoring Primes

- For relative primes, two numbers are relatively prime if the only shared integer factor between the two numbers is 1, i.e. the greatest common denominator is 1.
  - A couple of large primes for example are 3457631 and 4563413, which multiply together to be 15778598254603.

11.7.4 Assertions That Can Be Verified in Polynomial Time

More significant for the theoretical development of this section is that, for certain problems that are otherwise intractable, the *verification of a solution* can be a problem of Class **P**. We describe the details of this assertion below.

For example, the problem of finding the prime factorization of a given natural number N with n digits is believed to be of exponential complexity (see [SCL]). But in fact this is not known, as of this writing. More precisely, the complexity—according to the best currently known algorithm—is about of size $10^{\sqrt{n \ln n}}$, which means that the computation would take about that many steps. But the verification procedure is of polynomial complexity: If N is given and a putative factorization $p_1, \ldots, p_k$ is given, then it is obviously (just by inspection of the rules of arithmetic) a polynomial-time problem to calculate $p_1 \cdot p_2 \cdots p_k$ and verify (or not) that it equals N.

The problem of factoring a large integer is believed by all the experts to be of exponential complexity (that is, if the integer has n digits then it will take on the order of 2^n steps to find the prime factorization).[114] The celebrated RSA encryption method—one of the cutting-edge techniques in modern cryptography—is premised on this hypothesis. That is, if we could find a way to factor large integers in polynomial time, then RSA-encrypted messages could be rapidly decrypted.[115] At this time it is not known whether the factorization of large integers is a polynomial-time problem. The best algorithms that we have are exponential time algorithms.

Likewise, the subgraph problem is known to be of exponential complexity. But, if one is given a graph G, a subgraph K, and another graph H, then it is a problem of only polynomial complexity to confirm (or not) that H is graph-theoretically isomorphic to K.

The considerations in the last three paragraphs will play a decisive role in our development of the concept of "problems of class **NP**."

Krantz (2011) "The Proof is in the Pudding: The Changing Nature of Mathematical Proof "

The Breakthrough Involved Something That Seemed an Absolute Heresy

- That is a public key
  - Previously, the same key that scrambled a message would also be the instrument that descrambled it. Keeping those keys secret was difficult:
    - The very tools that eavesdroppers lusted after, the decryption keys, had to be passed from one person to another, and then exist in two places, dramatically increasing the chances of compromise.
  - But under Whit Diffie's design, the public key that actually performed the encryption didn’t have to be a secret at all
    - It was allowed to be public and broadcasted
- Specifically, instead of keeping everything secret, all encrypted cipher texts are presumed to somehow be public anyway, as are functional form and spaces used for encryption, but of course not the secret message itself, or related aspects of the system such as the private key that enable secrecy.

Equivalent Mathematical Spaces: Some Algebra

- Algebra defines addition and multiplication as binary operations on any two elements of a set of objects, a group G.
  - any two elements in G can be added by a well- defined operation, and similarly there is an operator for multiplication of any two elements. **When certain familiar properties are satisfied, then G is referred to a ring**.
    - For addition the operator must be associative, commutative, have an identity element (0), and have an inverse element.
    - For multiplication, the operator must be left and right distributive and associative, and in some cases commutative, have of an identity element (1), and have an inverse.

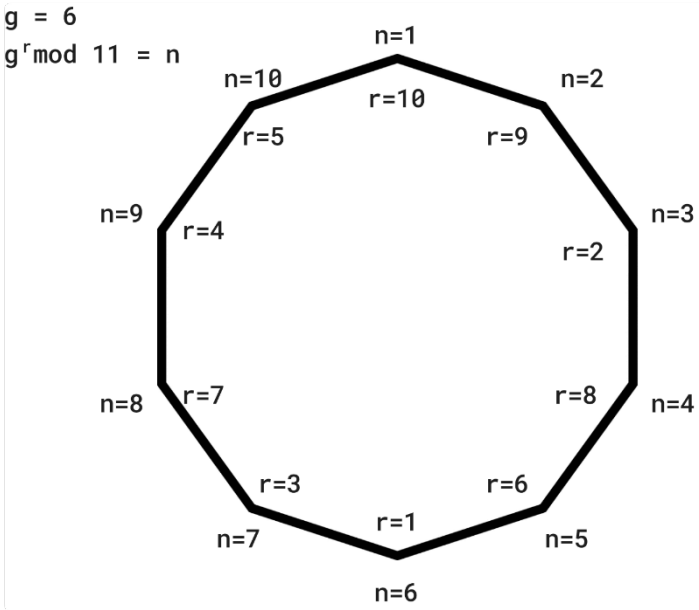
Rings

- One example of a ring
  - A set consisting of a finite number of integers from 1 to n, denoted Zn, with the proviso that addition of two integers is as usual except that if the sum is larger than n, one divides the sum by n and reports only the remainder.
    - This is denoted modulus n, or mod n.
- Multiplication
  - The number of elements in G is termed the order of the group G, and it is finite.
  - A generator g of group G is an element of G which when multiplied successively, under the multiplication operator, generates the entire space G. Such a multiplication operation repeated r times on an element h of G is represented in notation by taking element h to the specified power r, namely h^r.

Example of Cyclic Ring

- For cyclic rings, a good example is the ring where the order of G is 11, and $g = 6$ is the generator. g^r mod 11 is as follows:

$$r = 1 \rightarrow 6,\ r = 2 \rightarrow 3,\ r = 3 \rightarrow 7,\ r = 4 \rightarrow 9,\ r = 5 \rightarrow 10,$$
$$r = 6 \rightarrow 5,\ r = 7 \rightarrow 8,\ r = 8 \rightarrow 4,\ r = 9 \rightarrow 2,\ r = 10 \rightarrow 1$$



- In ElGamal cryptography, keeping the private key secret when the order of the cyclic group is small is impossible because with a small order, an attacker can easily brute-force calculate the private key by trying all possible values within the group order through a process called "baby-step giant-step" algorithm; therefore, it's crucial to choose a cyclic group with a very large order to ensure the security of the private key.

Key to Understanding Encryption

- when the repetition operation r is random and concealed, then with the modulus, outcomes from repeated operations put one back to a random and uninterpretable element of the finite set G. Rings with generators g are termed cyclic rings.
- Here r is the private key and the value h^r is the public key.
- The space G with its operations and modulus, order, and generator are understood and are public as well, in addition to the public key.
- The method of encryption is regarded as being in the public domain in modern cryptanalysis.

ElGamal Encryption: Incoming Messages

ElGamal encryption consists of three components: the key generator, the encryption algorithm, and the decryption algorithm.

[Key generation] The first party, Alice, generates a key pair as follows:

Generate an efficient description of a cyclic group G of order q with generator g. Let e represent the unit element of G (additive group).

Choose an integer x randomly from $\{1, \ldots, q-1\}$.

Compute $h := g^x$.

The public key consists of the values (G, q, g, h). Alice publishes this public key and retains as her private key x, which must be kept secret.

[Encryption] A second party, Bob, encrypts a message to Alice under her public key (G, q, g, h) as follows:

Map the message m to an element m of G using a reversible mapping function.

Choose an integer y randomly from $\{1, \ldots, q-1\}$.

Compute $s := h^y$. This is called the shared secret.

Compute $c_1 := g^y$.

Compute $c_2 := m \cdot s$.

Bob sends the ciphertext (c_1, c_2) to Alice.

[Decryption] Alice decrypts a ciphertext (c_1, c_2) with her private key x.

Compute $s := c_1^x$. Since $c_1 = g^y$, $c_1^x = g^{xy} = h^y$and thus it is the same shared secret that was used by Bob in encryption.

Compute s^{-1}, the inverse of s in the group G.

Compute $m := c_2 \cdot s^{-1}$. This calculation produces the original message m, because $c_2 := m \cdot s$; hence $c_2 s^{-1} = m \cdot s \cdot s^{-1}$.

Key Signatures, outgoing messages

- Signatures allows verifiability of the sender's identity and message content, hence commitment to being the author and having sent that message.
- The private key is used to generate a digital signature for an outgoing message, and such a signature can be verified by those who know the signer's corresponding public key.
  - Remarkably, the recipient can figure out who sent the message.
- More specifically, the receiver can verify the origin of the message (Authentication), the sender cannot falsely claim that they have not signed the message (Non-repudiation), and that the message has not been modified since it was signed (Integrity).
  - These are extremely useful properties for verifiability and commitment, all without the receiver ever knowing the private key.

Schnorr Signature Scheme: Outgoing Messages

Key generation has two phases. The first phase is a choice of algorithm parameters which may be shared between
different users of the system. The second phase computes a single key pair for one user.

Parameter generation

- Choose a key length N.
- Choose a N-bit prime number p
- Choose a cryptographic hash function H with output length L>N, only the leftmost N bits of the hash output are used.
- Choose a generator g<p of the cyclic group of integers modulo q (q is different from p !!)
- The algorithm parameters are (p,g). These parameters may be shared between users of the system.

Per-user keys

Given a set of parameters, the second phase computes the key pair for a single user:

- Choose an integer a randomly from $\{1, \ldots, p-2\}$.
- Compute y: = g^a mod p.

a is the private key and y is the public key.

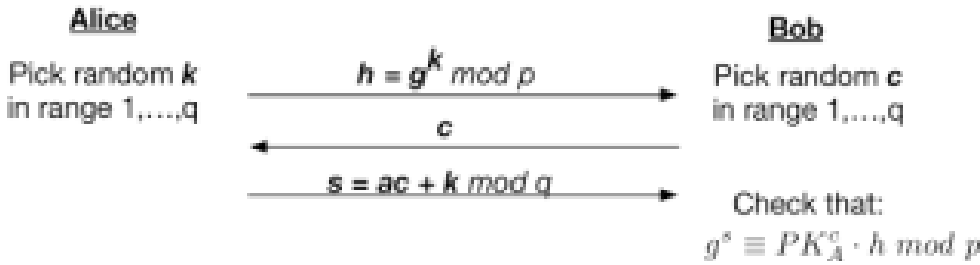
$$PK_A = g^a \ mod \ p \qquad SK_A = a$$

Signing

A message m is signed as follows:

- Choose an integer k randomly from $\{1, \ldots, q\}$ with k relatively prime to p−1.
- Compute h: = g^k mod p
- Compute s := $ac + k \ mod \ q$
- In the unlikely event that s=0 start again with a different random k.

The signature is (r,s).



Verifying a signature

$$\begin{aligned} g^s &\equiv PK_A^c \cdot h \mod p \\ g^{ac+k} &\equiv (g^a)^c \cdot g^k \mod p \\ g^{ac+k} &\equiv g^{ac+k} \mod p \end{aligned}$$

Bob can do that without knowing the secret key a !!

Correctness

The algorithm is correct in the sense that a signature generated with the signing algorithm will always be accepted by the verifier.

Examples of Rings: Polynomials

- To build such a ring (in crypto) we generally follow these steps:
  - Take coefficients of polynomials within a finite field (**F***q*)
    - Exemple of **F***q* : 𝔽2 [0,1] , 𝔽11 is [0,1,2,3,4,5,6,7,8,9,10] , 𝔽23 [0,1,2,3,4,5,6,7,8,9,10,11,12...22]
    - So all these coefficients are integers less than the degree *q of the* **F***q*

- *i.e. first choose* a complexity value of *n*, eg the highest coefficient power used.
- then generate *q* as 2^*n*−1. All polynomial operations will then be conducted with a modulus of *q* and where the largest coefficient value will be *q*−1
- then creates $a_i.\mathrm{x}$ which is a set of polynomial values:

$$\mathbf{A} = a_{n-1}x^{n-1} + \ldots + a_1 x + a_1 x^2 + a_0$$

- Then divide by Φ(x), which is x^n+1 in many cases:

$$\mathbf{A} = (a_{n-1}x^{n-1} + \ldots + a_1 x + a_1 x^2 + a_0) \div (x^n + 1)$$

In Python this is achieved with:

```
xN_1 = [1] + [0] * (n-1) + [1]
A = np.floor(p.polydiv(A,xN_1)[1])
```

General Principle:

How to Perform Computations and Transform Data in Encrypted Spaces Without Knowing the Underlying Meaning of the Inputs or Outputs?

Homomorphic Encryption

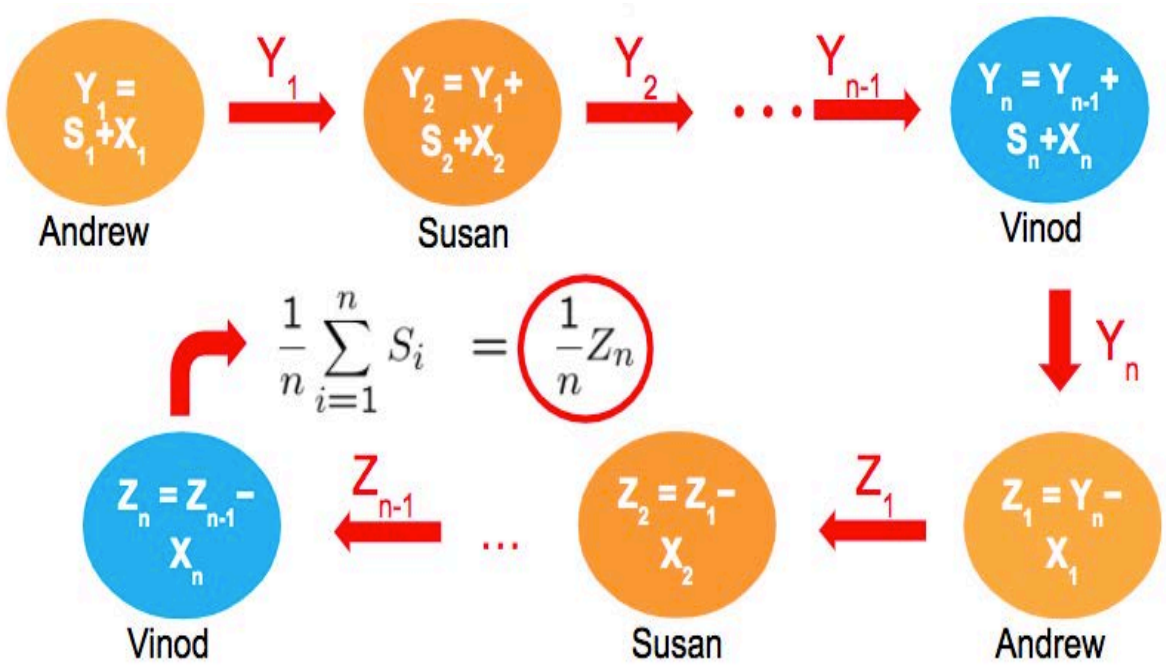
Homomorphic Encryption

- Homomorphic Encryption: the basic idea in mathematics is an isomorphism between two structures. A mapping from one to the other can be reversed by an inverse mapping.
- When the two structures are algebraic, isomorphic mappings are termed homomorphic if the mapping is bijective (1-1), with every element in the domain having a representation in the range and vice versa. Isomorphic mappings can preserve relationships across the two spaces.
  - Given a function f operating on messages m, we get $f(m)$
  - One can also do this operation f is an isomorphic space, namely $f(encrypted\ m) = encrypted[f(m)]$.
  - In the latter case: the encrypted inputs and outputs are obscure and make no sense. Yet one can recover the answer f(m) by $deciphering\ \{[f\ (encrypted\ m)]\}$.
- El Gamal is an example of homomorphic encryption

Multiparty Computation

- On the right **s private data** *(ex line items on financial accounts)*

- x are random numbers chosen by each participant *(the x's can be seen as each agent encrypting his secret and then all agents collectively decrypting at the end)*



- NB: this back and forth *(2.n messages sent for one computation among n agents in this example)* is slow if every message sent has to be "validated" by **consensus**!

Summary and Generalizing

- HE is for single agents whereas MPC is for multiple, though a bit more limited, as function f must be linear.
- Under MPC, the underlying messages before encryption, need never be revealed.

The key property of HE is that: **f[Enc(m)] = Enc[f(m)]**

Equivalently, **Decipher[f(Enc(m))] = f(m)**

Further, with ***distributivity*** we have flexibility in navigating between the "normal" space and the encrypted space. This allows us to perform a series of **composed transitive functions g on top of encrypted messages**:

$$g[\,Enc(\,f(m)\,)\,] = g[\;f(\,Enc(m)\,)\,] = Enc[\,g(f(m))\,]$$

Equivalently,

$$Decipher[\,g[\,Enc(\,f(m)\,)]\,] = Decipher\,[\,Enc(\,g(f(m))\,)] = g(f(m))$$

Zero Knowledge Proof

- How a prover can convince a verifier S/he knows something without ever revealing what that knowledge: **zero knowledge proof**
- A Zero Knowledge Proof :
  - It can be needed when we have pre-existing accounts on ledgers and wish to keep some secrets, such as the transfer.
  - The idea is for one party, the prover, to convince another, the verifier, that s/he knows something without ever revealing anything about what that knowledge is.
- The concern is not that the prover will deceive the verifier but the verifier will learn something and leak or misuse it.
  - e.g., a user as prover is given a password, and is subsequently hashed. Then when the password is used in the future, to verify s/he is the correct user, who knows the password, the server as verifier re-enters the password and compares the hashes. If the hashes match, then the user has proved to the verifier that s/he is who s/he says she is. **However, in this case the server as verifier has actually learned the underlying secret,** the password, the worst possible outcome if the npromised.

First Example of Zero Knowledge Proof

- Suppose there are two pens which seem identical to the outside world, include the verifier, but in fact have a subtle difference only the prover knows and he claims this knowledge.
- The two pens are given to the verifier, one claimed to have special concealed markings, designated, and the other not.
- Though they appear identical to the verifier, she keeps track and scrambles them nevertheless behind her back. When shown to the prover, left hand or right hand, the prover picks the designated one.
- Repeat the experiment to avoid prover's right guess because of luck. If the prover really knows the difference, the verifier is asymptotically convinced the prover has this knowledge.
- Here the verifier never knows actually knows which pen is which, zero knowledge.

Second Example of Zero Knowledge

- A network of nodes, as circles, some of which are connected to each other with edges, as lines. **The three-color challenge is to paint all the nodes of the network so that no two nodes connected by an edge share the same color.**
- A prover claims to have a solution, and needs to convince a verifier without revealing anything about the solution.
- The verifier picks one edge, and the two connected nodes are shown, which must have distinct colors if the prover actually has a solution and is trying to convince the verifier.
- The experiment is repeated, except that all the nodes are recolored secretly in advance by the prover.
- Data from the first and second experiment cannot be connected, so what is happening is a series of independent trials. **As the three color problem is NP-complete it maps into all sorts of interesting real word applications.**

ex of ZK on ledger (with homomorphic addition)

ID	Asset	From	To	Amount
1	€	Depositor	Goldman Sachs	30M
2	€	Goldman Sachs	JP Morgan	comm(10M)
3	€	JP Morgan	Barclays	comm(1M)
4	€	JP Morgan	Barclays	comm(2M)

Pedersen commitments **(a scheme very close to El Gamal ! –the additive version)**

Bank creates $\texttt{comm(v)} = g^v h^r$

The zk proof

For a transfer transaction in row m, each entry i contains the following items:

- **Commitment** (cm_i): $(g^{v_i} h^{r_i})$ a Pedersen commitment to the value we are transferring.
- **Proof of Balance** (π^{B}): a zero-knowledge proof asserting that the committed values satisfy $\sum_{k=1}^{n} v_k = 0$.

This can be proved by giving to the verifier r1+r2 from comm(v1) and comm(v2)

ZkLedger: Privacy-Preserving Auditing for Distributed Ledgers, Class slide 51

- **Cryptocracy building blocks: Neha Narula et al (2018)**
  - Commitments schemes. To protect their privacy, participant banks do not broadcast payment details, such as the transaction amount. Instead the banks post hiding commitments to the append-only ledger
  - zkLedger uses Pedersen commitments. Let $\mathbb{G}$ be a cyclic group with $s = |\mathbb{G}|$ elements, and let g and h be two random generators of $\mathbb{G}$. Then a Pedersen commitment to an integer $v \in \{0,1, \dots, s-1\}$ is formed as follows: pick commitment randomness r, and return the commitment $cm \coloneqq COMM(v,r) = g^v h^r$.
  - A very useful property of Pedersen commitments is that they are additively homomorphic. If cm_1 and cm_2 are two commitments to values v_1 and v_2 using commitment randomness r_1 and r_2, respectively, then $cm \coloneqq cm_1 \cdot cm_2$ is a commitment to $v_1 + v_1$ using randomenss $r_1 + r_2$, as $cm = (g^{v_1} h^{r_1}) \cdot (g^{v_2} h^{r_2}) = g^{v_1+v_2} h^{r_1+r_2}$.
  - Though keys *r* are private, the sum of keys can be be given to third party auditor to verify sum of values *v*.

Encryption as Utilized in Bitcoin

Merkle Trees, Block ID Hashes in proof of work

Block Structure

Each block consists of a ***header***, a transaction count, and the list of transactions

- The header contains information about the block, such as the previous block ID, the timestamp, a nonce, and difficulty

- Ethereum block headers are much more complex to properly track state across contract execution
  - State tree root
  - Total gas used for the whole block
  - etc.

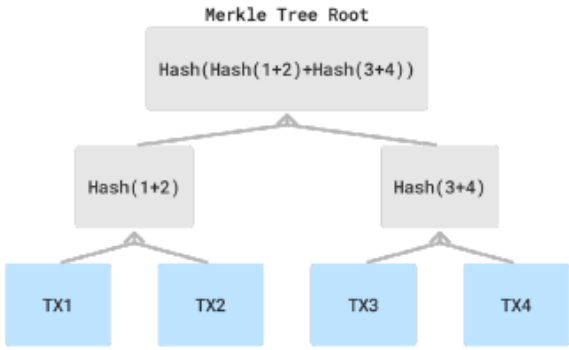
Bitcoin Block Structure

Block ID	Block ID	256 bits
Previous BID	Block ID of the preceding block	256 bits
Merkle Root	Merkle Tree hash of transactions in the block, for verification	256 bits
Timestamp	Time the block was created	32 bits
Nonce	Unique number used for cryptographic hashing	32 bits
Version	Block format version	32 bits
Difficulty	The difficulty rating for proposing this block	32 bits
Transaction Count	The number of transactions in the block	Variable size
Transactions	List of transactions in the block	Variable size

Block Structure

Merkle Trees, or Hash Trees, are used to validate the transactions in each block

- Merkle trees use a combination of hashes of the transactions to create an efficient and easily verifiable summary of transactions in the block
- The leafs of the tree are the transactions, and each node consists of the hash of it's two children
- Working back to the top, the root node is the combination of all transactions
- Since changing any transaction would change the root, storing the root in the header makes the transactions tamper proof



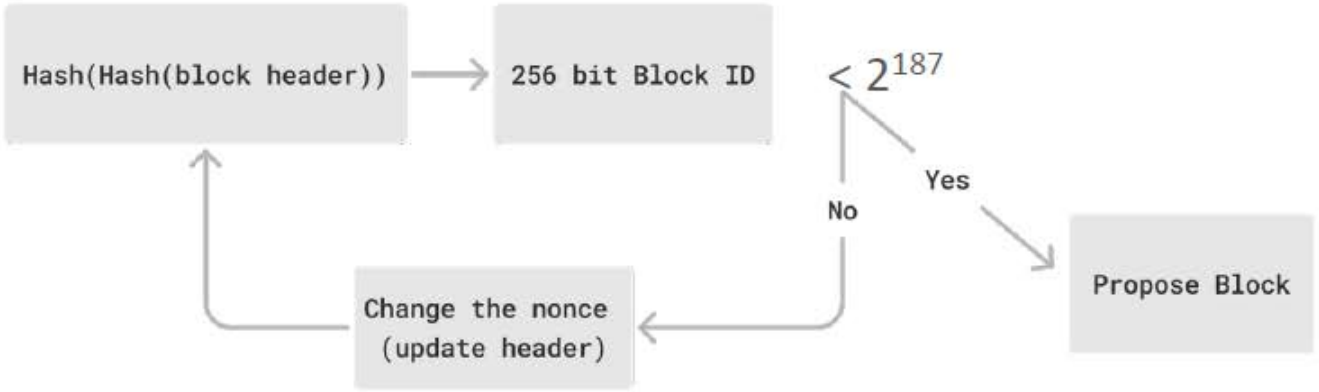
Creating the block ID

- The ID of each block is computed by double hashing the entire rest of the block header
  - The double hashing process maps the block header to a random 256 bit number, the new ID
  - The nonce is a unique number used to change the ID. Modifying the nonce (or any other part of the header) will result in a different random doubled hashed number

```
Hash(Hash(block header)) -> 256 bit Block ID
```

Block ID in proof of work

- Proof of work is a consensus mechanism that limits block proposals using computational resources
- For a block to be valid, the ID must be smaller than a constant number (ex. 2^{187})
- The likelihood of a random 256 bit number ($0 < ID < 2^{256}$) being valid is very small, so proposers change the nonce and retry until a valid ID is found

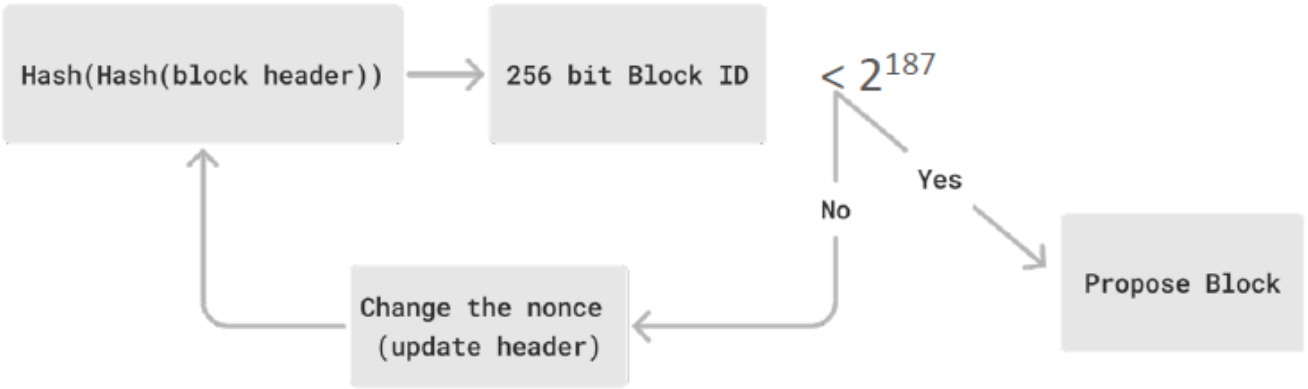


Hashes of Block Headers

- The hash of the header is exactly 256 bits long, in binary.
- However, each bit is a 1 or 0, so it is just a large number where every index i in the number represents adding 2^i to the sum.
- If every bit was 1, that number would be 1 + 2 + 4 + 8 + … = 2^256 -1, but if every bit was 0, the number would be 0, so essentially the hash outputs a random number.
- Example, if the first 187 bits happen to be 0, then the number is less than 2^69 and it's valid.
- Since it is random, that happens very infrequently.
- [Not sure where this slide goes? ??]

Proof of Work Continued

- The more computational resources a node devotes to the blockchain, the more tries that node makes to propose, the more successful proposals, thus increasing their influence
- To reward nodes for proposing blocks, any successful proposal can include an extra transaction to themselves, gifting free currency



Summarizing the State of Encryption