**ZK Core Program - Homework Week 1**

1. **Symmetric vs. Asymmetric Encryption**: What are the key differences between symmetric and asymmetric encryption? Provide a practical use case for each.

Answer :

* **Simple Explanation Symmetric Encryption (One Key)**: Uses the same key to lock (encrypt) and unlock (decrypt) data. Fast, but you must secretly share the key (risky if intercepted). Example: Like a physical key that both sender and receiver use. **→usecase :** Encrypting large files (e.g., ZIP passwords, hard drive encryption).
* **Asymmetric Encryption (Two Keys)**: Uses a public key (shared with everyone) and a private key (kept secret). Public key locks data, private key unlocks it. Slower, but no need to share secrets. Example: Like a mailbox—anyone can drop a letter in (public key), but only you can open it (private key). **→usecase :** Secure emails, Bitcoin transactions, or sending secret keys safely.

1. **Public-Key Cryptography and Key Exchange Protocols**: How can the Diffie-Hellman protocol enhance security in a messaging application

Answer :

* This protocol is significant as it anables secure communication over insecure channels by allowing two parties to generate a shared secret key, which can then be used for encryption and decryption of messages. It helps prevents eavesdropping because: Even if attackers see the public keys, they can’t compute the shared secret easily.
* Used in: Signal Protocol (WhatsApp, Signal). TLS/SSL (HTTPS secure browsing).
* Example of how it works: Alice and Bob agree on a public **prime number (p)** and **base number (g)**. Each picks a **private number (a, b)**. They compute: Alice sends: A = g^a mod p Bob sends: B = g^b mod p Shared secret = B^a mod p = A^b mod p (same result).

1. **Hash Functions**: What features make SHA-256 and Poseidon good hash functions for ensuring data integrity? Mention one unique advantage of Poseidon.

Answer :

* Deterministic (same input → same output).
* Pre-image resistance (hard to reverse).
* Collision resistance (hard to find two inputs with the same hash).
* Avalanche effect (small input change → completely different hash).
* SHA-256: Widely used in Bitcoin and TLS. Computationally secure but slow in zero-knowledge proofs (ZKPs).
* Poseidon (Unique Advantage): Unlike traditional hash functions like SHA-256, Poseidon is optimized for operations within prime fields, a common setting for ZK proofs, minimizing multiplicative operations and constraints. This leads to faster computations and smaller circuit sizes, making it a valuable tool for applications requiring privacy-preserving computations.

1. **Merkle Trees**: Explain how Merkle trees can help verify data in a large database efficiently.

Answer :

* A Merkle tree is a core component of blockchain and cryptography. It's a binary tree filled with cryptographic hashes. This structure enables efficient and secure verification of the contents of large data structures.
* Merkle trees enable efficient data verification in large databases by using a tree-like structure of cryptographic hashes. Each leaf node represents a hash of a data block, and parent nodes are hashes of their children. This allows for quick verification of data integrity and inclusion proofs, as only a small portion of the tree needs to be traversed to confirm a specific data point.

1. **Cryptographic Commitments**: How can Pedersen Commitments be used in a blockchain protocol to maintain transaction privacy?

Answer :

* Cryptographic commitments are essential in cryptography and blockchain technology as they allow for selective hiding and revealing of information. This feature ensures data privacy while still enabling verification processes.
* It helps achieve secure and efficient verification of transactions in blockchain protocols. In such contexts, sensitive information, such as transaction details or user identities, is hidden while revealing others for the verifier to authenticate the transactions.
* A Pedersen Commitment has two key properties - **Hiding**: The commitment reveals nothing about the secret value. **Binding**: The committer cannot change the secret after committing. It relies on hard mathematical problems (discrete logarithms) to achieve these properties.
* The commitment is computed as:

| Factor | *C* | *m* | *r* | *g,h* | *p* |
| --- | --- | --- | --- | --- | --- |
| Use: | Commitment value (sent to the verifier). | Secret message (e.g., a transaction amount). | Random blinding factor (kept secret). | Public generators of a cryptographic group (e.g., elliptic curve points). | A large prime number (group order). |

* It is secure because it **hiding property** - *C* looks random because of *r* , Even if *m* is known,  *r* makes it impossible to link *C* to *m*. And **binding property** to find (*m’,r*’) such that ***g^m.h^r=g^m’.h^r’*** requires solving : ***m + s.r == m’ + s.r’ (mod p-1***) and this is computationally infeasible which means discrete log problem

1. **Digital Signatures**: How can you verify the authenticity of a digitally signed document?

Answer :

* Digital signatures ensure the integrity and authenticity of digital messages or documents. By providing a means to verify the origin and confirm that the content has not been altered, digital signatures play a pivotal role in maintaining trust in digital communications.
* In PKC (Public Key Cryptography), anyone can encrypt their message with the receiver's public key, and only the receiver can decrypt the message with their private key. In digital signatures, on the other hand, if a signer generates a signature for a message using their private key, anyone can validate it using the signer's public key. Therefore, the message of the signature is made public, which distinguishes it from cryptographic commitments.
* Here are the verification steps:

| **Step** | **How it works** |
| --- | --- |
| 1 | Obtain Public Key: From a trusted source (e.g., SSL certificate, Bitcoin address). |
| 2 | Hash the Document: Generate a hash of the original message. |
| 3 | Decrypt Signature: Use the signer’s public key to decrypt the signature → reveals the signed hash |
| 4 | Compare Hashes: If the decrypted hash matches the document’s hash, the signature is valid. |

**THANK YOU !**