

# Blockchain Exercise

# Asymmetric Encryption

- Private key
- Public key
- Digital Signature Algorithm (DSA)

# Private key

vs

# Public key

- Available to its owner
- Also known as signing key
- Or SK

- Available to everyone
- Also known as verifying key
- Or VK



# Confidentiality VS Digital Signature Algorithm (DSA)

- Confidentiality:
  - Sender:
    - `m= 'msg'`
    - `M= encrypt(m, receiver.vk)`
    - `Send(M)`
  - Receiver:
    - `m= decrypt(M, self.sk)`
- DSA:
  - Sender:
    - `signature= encrypt(m, self.sk)`
    - `Send(signature, m)`
  - Receiver:
    - `verified= signature == decrypt(m, sender.vk)`


# Using Oracles to make API calls in Solidity

- An oracle is a trusted or decentralized entity that serves as a bridge between smart contracts on a blockchain and external data sources.
- It provides blockchain-based applications with access to real-world information, such as market prices, weather data, or any off-chain data.
- Oracles ensure that this data is reliable, secure, and tamper-proof, enabling smart contracts to make informed decisions and automate actions based on external events or data.



# Using Oracles to make API calls in Solidity is essential for several reasons:

1. **External Data Integration:** Smart contracts on the Ethereum blockchain are isolated from external data sources, making it impossible to interact with external systems or fetch real-world data. Oracles bridge this gap by providing a way for smart contracts to access off-chain information, such as prices, weather data, or sports scores.
2. **Decentralized Trust:** Oracles can be decentralized and trustless, ensuring that the data they provide is reliable and tamper-proof. Multiple oracles can be used to fetch the same data, and a consensus mechanism can be applied to ensure data accuracy.

- 
3. **Real-World Use Cases:** Many blockchain applications, like DeFi (Decentralized Finance) projects, require up-to-date market prices, and IoT (Internet of Things) smart contracts may need real-time sensor data. Oracles are indispensable for these real-world use cases.
  4. **Smart Contract Automation:** Oracles enable smart contracts to react autonomously to external events or data changes. For example, a smart contract could automatically execute a trade when a specific price condition is met.



5. **Trustworthiness and Security:** By using reputable oracles and implementing proper security practices, you can reduce the risks associated with unreliable or malicious data sources. Oracles allow you to leverage the security of the blockchain while still accessing external data.

However, it's crucial to choose reliable and secure oracles, implement safeguards to handle erroneous data, and be aware of potential attack vectors when integrating oracles into your smart contracts.



# Event concept in Solidity

- In Solidity, an "event" is a way to log and emit information from a smart contract to the Ethereum blockchain.
- Events are an essential part of the Ethereum ecosystem, as they allow external applications, like decentralized applications (DApps) or other smart contracts, to listen for and react to specific occurrences within a contract.

Here's how events work:

# Declaration

- You declare an event in a Solidity contract using the `event` keyword. For example:

```
event ItemPurchased(address indexed buyer, string item, uint256 price);
```



# Logging

- Inside the contract functions, you use the `emit` keyword to log an event. For instance:

```
function purchaseItem(string memory _item, uint256 _price) public {  
    // ... (contract logic)  
    emit ItemPurchased(msg.sender, _item, _price);  
}
```

# Consumption

- External applications or other smart contracts can subscribe to these events and listen for them. They can react or gather data based on the information emitted by the events.

1. Events are crucial for transparency and interoperability in the Ethereum ecosystem. They provide a way to notify the world about state changes within a contract, enabling various use cases such as tracking transactions, monitoring token transfers, and triggering actions in response to specific events.

## Summary



- 
2. The `indexed` keyword, as shown in the event declaration, allows efficient filtering and searching when listening for events. It is typically used with address types to optimize event log retrieval.

## Summary

3. Events are not directly readable on the blockchain; you would need to interact with a blockchain explorer and use a web3 library like Web3.js or ethers.js to access and interpret event data through that explorer.

## Summary