Q. Write a Solidity Smart Contract for a Non-Fungible Token (NFT) which can able to perform Identity Verification and Authentication as they can contain unique information about an individual that is Cryptographically secure.

Ans. Creating a Solidity smart contract for an NFT with identity verification and authentication is an interesting use case. Here are the steps involved using standard ERC-721 for NFTs with additional functions for identity verification:

Steps:
1. ERC-721 Standard
2. Identity Information
3. Verification
4. Authentication

Smart Contract (Solidity)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract IdentityNFT is ERC721, Ownable {
    struct Identity {
        string name;
        string dob; // Date of Birth (format: YYYY-MM-DD)
        string metadataHash; // Cryptographic hash of personal information (could include address, government ID, etc.)
        bool verified; // Whether the identity is verified
    }

    // Mapping token ID to Identity
    mapping(uint256 => Identity) private _tokenIdentity;

    // Event for identity verification
    event IdentityVerified(uint256 tokenId, bool verified);
```

```
    // Constructor to initialize the NFT collection
    constructor() ERC721("IdentityNFT", "IDNFT") {}

    // Function to mint a new NFT and associate it with identity information
    function mintIdentityNFT(
        address recipient,
        uint256 tokenId,
        string memory name,
        string memory dob,
        string memory metadataHash
    ) public onlyOwner {
        _mint(recipient, tokenId);
        _tokenIdentity[tokenId] = Identity({
            name: name,
            dob: dob,
            metadataHash: metadataHash,
            verified: false
        });
    }

    // Function to get identity information
    function getIdentity(uint256 tokenId) public view returns (string memory name, string
memory dob, string memory metadataHash, bool verified) {
        require(_exists(tokenId), "Token ID does not exist");
        Identity memory identity = _tokenIdentity[tokenId];
        return (identity.name, identity.dob, identity.metadataHash, identity.verified);
    }

    // Function to verify the identity by checking the hash
    function verifyIdentity(uint256 tokenId, string memory providedHash) public onlyOwner {
        require(_exists(tokenId), "Token ID does not exist");
        Identity storage identity = _tokenIdentity[tokenId];
        require(keccak256(abi.encodePacked(providedHash)) ==
keccak256(abi.encodePacked(identity.metadataHash)), "Invalid hash provided");

        Identity.verified = true:
        emit IdentityVerified(tokenID, true);
    }
```

```
    // Function to authenticate the identity (can be called by anyone with the tokenId)
    function authenticateIdentity(uint256 tokenId) public view returns (bool) {
        require(_exists(tokenId), “Token ID does not exist”);
        Identity memory identity = _tokenIdentity[tokenId];
        return identity.verified;
    }
}
```

Key Components:

1. **ERC-721 Standard:** This is the base for creating an NFT. The mintIdentityNFT function allows the owner to mint a token and assign identity information to it.
2. **Identity Struct:** Contains the user’s name, dob, metadataHash and a verified flag.
3. **Verification:** The verifyIdentity function allows the owner to verify the identity by comparing the hash of provided data with the stored metadata hash. If they match, the identity is marked as verified.
4. **Authentication:** The authenticateIdentity function can be used to check whether the identity has been verified.