Enabling Secure and Privacy Preserving Identity Management via Smart Contract

Yaoqing Liu
liu@clarkson.edu
Clarkson University

Guchuan Sun
sung@clarkson.edu
Clarkson University

Stephanie Schuckers
sschucke@clarkson.edu
Clarkson University

***Abstract*—Biometrics have been used increasingly heavily for identity authentication in many critical public services, such as border passes or security check points. However, traditional biometrics-based identity management systems collect and store personal biometrical data in a centralized server or database, and an individual has no control over how her biometrics will be used for what purpose. Such kind of systems can result in serious security and privacy issues for sensitive personal data. In this paper, we design a novel approach to leveraging biometrics and blockchain/smart contract to enable secure and privacy preserving identity management. The basic idea is to use blockchain to store an authority's attestation and the transformed value of an individual's biometrics. The stored data on the blockchain is then controlled by smart contracts which define various access control policies, e.g., access parties, access times, etc. The owner of the biometrical data can flexibly change the access control policies through a white list, a timer and other methods to any identity verifiers. We used the well-known Ethereum platform to implement the proposed approach and tested the effectiveness as well as the flexibility of various access control policies.**

I. Introduction

Identity is the distinguishing attribute of an individual. It is widely referred to as the sum of attributes, such as biometrics, height, and birthdate, or the designed attributes, e.g., driver licenses and passports, associated to a person. It is not a trivial job to manage individuals' credentials that connected to their Identities, since the attributes may change or be lost over time. Currently, besides biometrics-based identity management [1], there are three primary identity management practices in the field: physical credentials, digital credentials and cryptographic credentials.

Government issued credentials, such as passports, birth certificates, and driver's licenses, belong to physical credentials. Their copies are kept in a centralized authority and protected by digital storage. An individual needs to show their hard copies to prove their identities first before receiving the services, such as air travels and border passes. The potential problems for this kind of identity management is that the physical forms can be lost, stolen or damaged, and even faked. The replacement incurs high costs as well. The second type of credentials are enterprise issued credentials, such as Facebook and Google accounts. When a user wants to prove herself to access an online service, she can just link the service to her digital credentials to be authenticated. This kind of credentials can facilitate many online services and become very popular recently. However, many critical public services cannot be supported by them, e.g., opening a bank account and security check. Other issues include the single point of failures and potential data breach risks. For instance, Facebook and Yahoo!'s user data had been breached and hundreds of millions of users' sensitive credentials were disclosed to the public [2]. The third type of credentials are cryptographic credentials. An individual requests a public/private key pair through Public Key Infrastructure (PKI). The user verifies herself through her digital signed document. However, such kind of applications are limited to electronic operations and the keys may be expired and lost as well. Also the requesting procedure is cumbersome, costly and the PKI infrastructure yields single point of failures.

The concerns for identity management include security, privacy, credential revoking, credential recovery, flexible credential access control, etc. Traditional approaches cannot meet all of the requirements well for a good identity management system. One of the main challenges is the security issue, e.g., single point of failure issue, because most of them use a centralized server or database to store the sensitive personal identity data. If the system was compromised, the services providing identity data will not be available to access, and the data risks being stolen and abused. Regarding privacy, current policies require keeping user data unaccessible by a third party without the user's permission [3], however, the implementing mechanisms cannot guarantee or enforce such kind of policies to be in effect particularly when the system was hacked. In terms of flexible access control over an individual's personal data, current practices do not offer effective mechanisms to realize, and the owner of their data can do nothing but trust and rely on the service providers or government agencies to not abuse their personal data. The situation worsens when the data pertain to the user's biometrical information. For instance, malicious users may be able to access the biometrical data and fake them for false identities [4]. Therefore, it is critically important for users themselves to control what they own and authorize who can access them.

To address these concerns and challenges, we explore the state-of-the-art biometrical identification and Blockchain technologies [5] to enable secure, privacy preserving, and flexible identity management. Biometrcis, such as fingerprints, face, voice etc., are something that an individual owns and they are always with the person and unique as well. Most of time, they are resist to loss and changes [6]. Combining advanced technologies in biometrical identification, identity

management, Blockchain and smart contract, we make the following contributions in the paper:

1) Framework: We designed and developed a new and novel identity management framework that spans the whole life cycle for managing an individual's biometrical data for authentication. It includes modules of identity collection, identity issuance, identity storage, identity transformation, identity access control, and identity verification. The identity collection and issuance are conducted by government-specified authorities, such as Department of Motor Vehicles (DMV). The identity storage uses personal or secured public repositories to protect information privacy, such as Dropbox or InterPlanetary File System (IPFS) [7]. The identity transformation employs well-known one-way hashing algorithms to ensure information security. The identity access control polices are enforced by public smart contract platform Ethereum. Finally, the identity verification can be executed by any party specified in the smart contract through a triple-way verification process.

2) Blockchain and Smart Contract: We built the framework based on the platform Ethereum, which is an open-source, public, blockchain-based distributed computing platform featuring smart contract functionality. The platform can guarantee the transparency and immunity of any state transitions via a modified version of Nakamoto consensus algorithm. We use the Ethereum platform as an anchor to provide transformed ground-truth identity information for an indiviudal. Meanwhile, the smart contract offers flexible programmable capabilities to define who at what time has the permission to access the ground-truth information.

3) Privacy Preserving: Our new identity management system features privacy preserving functionality for individuals' sensitive biometrical or other personal data. Traditionally, such kind of data are required to store in a centralized server or database for authorities to use as ground-truth information. In our new design, we do have to use ground-truth information to verify an individual's identity, however, this information is not the user's raw data and we only store encrypted transformed data in the blockchain. This ensures that the privacy of an user's data can be preserved.

4) Self Sovereign: In the new identity management system, personal data do not store in a centralized storage and cannot be used for any other purposes without the permission from owners. Users can register, retrieve and even revoke the data if they do not want to use them any more. In other words, the owners have the real ownership of their data in terms of where to store the data, how to share the data, who can access the data as well as when to retrieve the data.

II. Ethereum and Smart Contract

Our framework relies on the smart contract platform-Ethereum [8], a well-known blockchain application. In this section, we introduce the background of blockchain, including Bitcoin, hashing verification, decentralized structure, proof of work concept and the smart contract.

A. First Blockchain Application

Since 2009 BitCoin was proposed with the famous white paper by Satoshi [9], blockchain came into the public's view as a secure and distributed cryptocurrency. Bitcoin, as the first blockchain application, proposed a new model of network security and becomes the foundation of many blockchain-based applications, such Litecoin, Ethereum etc. The security and robustness of Bitcoin ecosystem are guaranteed by the following features:

1) Hashing Verification: Blockchain is not just a list of block objects, the information inside each block is well-designed such that everyone is able to verify the correctness of each block. Hash function is used in Blockchain, which uses one-way function to transform big chunks of data into a number with a fixed length. The return value of a hash function are unique and asymmetric, which means the results of hash functions are not predictable and a tiny change of the input will lead to a totally different result. As a result, when a new block is created, it hashes its previous block and records the result. Since the previous hash of Genesis Block (the first block) is well-known, it is easy to check the correctness of a blockchain by hashing each block, and comparing it with the "previous block's hash" value stored in the next one from the head all the way to the tail. Therefore, it is very difficult, if not impossible, to make any changes to the blockchain once a block is placed deep in the blockchain.

2) Proof of Work: Every miner wants to add a new block which contains many transactions so that the miner can be rewarded with a certain number of Bitcoins. But how can the minder win the reward while competing with other miners? In order to solve this problem, Bitcoin holds a "problem-solving competition". Every miner needs to find a hash input which leads to the result hash value smaller than a target value. The first one who successfully finds that value can add the reward transaction into the new block. The work taken in this competition is called Proof of Work (PoW), which reveals the calculation power included in "mining" the new block. Since hash function's result is not predictable, trial and error is the only way to obtain a desirable output, which takes longer time when the target value becomes smaller. PoW mechanism sets a high barrier to those who want to make up a longer chain from the start and attempt to replace the valid chain with a invalid one.

3) Decentralized Structure: Blockchain is a unique data structure which is shared among all of the participated nodes. When a new block is created by a miner, it is broadcast to the network and all other miners will verify its correctness, which contains the correctness of transactions, hash values and something other attributes. It is only when more than 50% of the miner nodes consider this block valid can the block be appended to the chain. All of the miners trust in the longest chain, which indicates a higher calculation power. As a result, all of the miners will eventually have the same blockchain. If a malicious user wants to make up a block and let it become valid, he must generate more than 50% hash power to compete with other nodes, which is extremely hard.

4) Keys and Addresses: The security of a user's access is guaranteed by an asymmetric crypto algorithm, called Elliptic Curve Digital Signature Algorithm, which is similar to RSA. This algorithm generates a key pair for every user. One key is called Public Key which is available to the public and the other is called Private Key, which is kept secret by the user. The public key actually is derived from the private key. However, this process is only one way. With a public key, it is theoretically impossible to find the private key. The user can encrypt a file or message with one key and use the other key to decrypt the result. This transaction verification acts in this way: The user transforms their public keys to their user's receiving addresses. Every transaction indicates the BitCoin comes from one address to another. When a user wants to make a spending, he needs to prove that the balance of the transaction belongs to him. Since his public key is known to others and he does not want to share his private key, he signs the spending transactions with his private key and broadcasts the signature to the network. Miners decrypts his signature through the public key to check if the decrypted content is the same as the original transaction. Note that the address itself is not the public key, it is derived from the public way using a one-way hash function.

B. Ethereum and Smart Contract

As a new generation of blockchain applications, Ethereum inherits the same principles of Bitcoin: it still uses hash function to verify the correctness of each block. Since Ethereum transactions contain not only Ether transferring but Smart Contract operations, there is more information needed to be recorded into blocks. As a result, the blocks are generated more frequently than Bitcoin. The proof-of-work in Ethereum is similar to Bitcoin. The address generation process and decentralized structure are similar to Bitcoin as well.

The most distinguishing difference between Bitcoin and Ethereum is that Ethereum has Smart Contract functionality. Smart Contract enables users with programming capabilities in Ethereum and users can flexibly define rules and policies. It implements user-defined protocols that can be executed into Ethereum Virtual Machine(EVM). Users can program smart contracts using Solidity, a special programming language to define various variable, classes and methods, like programming in an object-oriented programming environment. After a smart contract is created, it can be deployed to Ethereum platform by a user with some virtual currency in the form of gas. A deployed contract has a unique address that is publicly available for all users. Authorized users may access or even use the contract's methods through the address to change the state of the smart contract.

III. Framework overview

As shown in Figure 1, the life cycle of a user's authentication consists of three primary stages: user identity registration and smart contract construction, access control management via smart contract, and identity information retrieval and verification. We will walk through these stages in detail as follows.

A. Registration and Smart Contract Construction - Steps 1-5

To verify a user's identity, a verifier needs to use ground-truth or authority's information to match against. Without such kind of information, it is almost impossible to tell the true identity of an individual. Therefore, the first stage of the framework include steps to register a user's ground-truth information to a secure distributed system, or so called blockchain through an authority party, and then to construct a smart contract for the user (data owner) to manage access control policies. In the following context, we use the user's fingerprint as the user's biometrical data, Department of Motor Vehicle (DMV) as an authority party, and IPFS as the user's personal storage.

First of all, the user takes government issued documents, such as passport or driver license, to a local DMV to register her personal information, such as name, address etc (Step 1). Meanwhile, her fingerprint will be collected in the form of an image. Rather than storing the raw fingerprint image to the blockchain, the user stores them in a personal repository, such as IPFS [7] (Step 2). The image can be encrypted for another layer of security if necessary. The IPFS returns a hash ID as a retrieval key to the user for the fingerprint image (Step 3).

Second, DMV combines the returned hash ID and the user's non-sensitive information into a single data block, then hash the entire data block with a well-known hash algorithm, e.g., SHA256. Since hash function is a one-way calculation process, it is impossible to infer the content of the data block through the hash value itself. Afterwards, DMV signs the hash value with their private key. Note that the corresponding public key of DMV should be public, verifiable and traceable through PKI infrastructure. This way, a verifier can check the validity of DMV (issuer) through the signature.

Third, DMV integrates the signed attestation into a smart contract via Ethereum platform (Steps 4-5). The smart contract defines that the owner of the data is the registered user (with a smart contract user address), and the owner can later control the access policies to the data. Note that both DMV and the user should have their own user accounts of smart contract for these operations. In other words, the owner of the smart contract is DMV, however, only the user can control how to access the content of the smart contract. Furthermore, the authority can invoke or disable the smart contract at any time when necessary. For instance, DMV can disable the old smart contract and deploy a new one when the user has updated her biometrical information.

B. Access Control Management via Smart Contract - Step 7

After the construction of the smart contract by DMV, the user should have the privilege to access the smart contract via her smart contract user account. While communicating with the smart contract, the user account address will be verified by the smart contract every time when the user accesses it. Once the user is able to enter the smart contract, she has

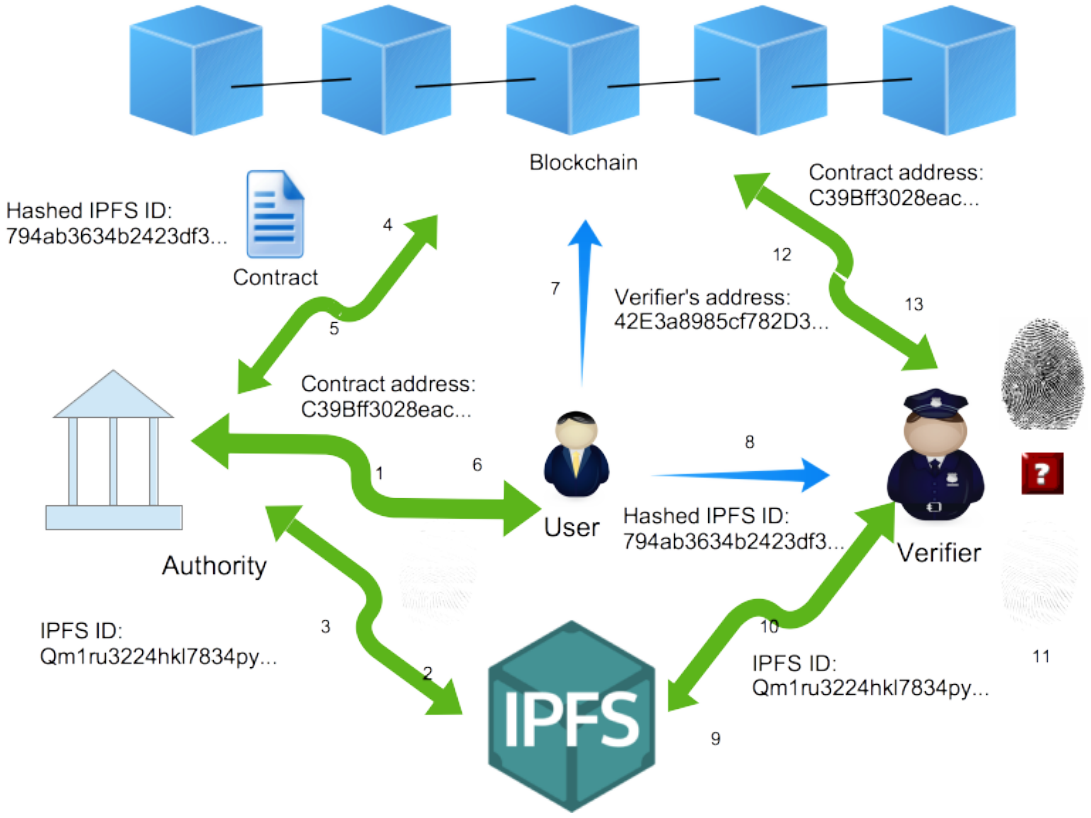

Fig. 1: Life Cycle of a User's Authentication

the privilege to program a whitelist that specifies what parties (which smart contract user accounts) can read the data from the smart contract, and how long the permission will be given. For example, multiple verifiers can be added into the whitelist, e.g., border patrol offices, security check points and other critical public service agencies that require biometrical data verification.

In our project, we have implemented two functionalities: who and when. The capabilities can be further enhanced, such as what information can be accessed by which party if there are multiple pieces of information available in the smart contract. Another example is that a verifier can access the information no more than a specified number of times.

C. User Identity Retrieval and Verification - Steps 8-13

After the first two stages, the user's transformed information has been stored in the smart contract and the user has also specified who and when can retrieve the information. This third stage describes how the user's identity is verified by a verifier at a security check point, for instance.

First of all, the verifier does not have any information about the user. The user needs to provide two pieces of information to the verifier to pass the verification of her identity. The first part is her original fingerprint image that was captured during the initial registration stage at DMV, and the second part is to collect her fingerprint information again on the scene (Steps 8 and 9).

To obtain the original fingerprint image, the user needs to use her IPFS hash ID to retrieve from the IPFS storage system and then decrypt it if the image was encrypted before (Steps 8, 9, and 10). This original image will then be compared with the newly collected fingerprint image with any state-of-the-art fingerprint comparison algorithm (Step 11). This step is imperative to verify that the user has its own fingerprint image (does not use others) showed to the verifier.

Although it can verify that the fingerprint image was owned by the user, it still cannot prove that other identity information such as name, or address of the user. The next step is for the verifier to retrieve DMV's attestation about the user's information (hash value of the IPFS hash ID and other personal information) from blockchain/smart contract (Steps 12 and 13). Since the verifier is within the whitelist of the smart contract, the DMV's attestation can be retrieved with the verifier's smart contract address. Afterwards, the verifier needs to check if the attestation was signed by DMV with its public key. If this step passes, it indicates the information in the smart contract is true.

Based on this, the verifier will combine the user information and the encrypted IPFS hash into a data block, and conduct a hash operation using the same hash function as DMV did initially. The newly generated hash value should exactly match the one obtained from the smart contract. If this step passes, the verifier can confirm that the user is the same person who claimed to be in DMV when she registered her identity. Here

we use a three-party verification: ground-truth info provided by DMV, the user's claim and her biometrical check for security and privacy preserving identity verification.

IV. Implementation

We use Ethereum (0.11.1) and Go-ethereum (1.8.21) as our smart contract platform. We build the smart contract with Solidity (0.4.24) programming language. We use Python 3.6.4 programs to simulate different parties, including authorities, users, and verifiers. The python programs connect to Ethereum to deploy smart contract for an authority, to authorize permissions for a user, and to retrieve the user's data for an verifier. We employ IPFS (0.4.18) to store the user's personal biometrical information (encrypted if necessary). The python programs also communicate with IPFS to upload or retrieve the user's data via a hash ID. In the following sections, we describe the detailed algorithms and implementation.

A. Smart Contract

We introduce the smart contract template as shown in Algorithm 1, which contains a $constructor$ function to initiate a smart contract, an $authorize$ function for an owner to specify access control policies, an $access$ function for a verifier to retrieve information, a $revoke$ function for an authority to disable the smart contract, and an $ObtainAuthority$ function for permissioned parties to verify that the ownership of the smart contract.

The smart contract includes five variables: the owner of the smart contract, the biometrical hashing data of the owner, the smart contract address of the authority, a flag to indicate if the smart contract is active and enabled, and a whitelist to specify who can access the smart contract. In the $contructor$ function, the first step requires that only the party that has the public smart contract account address can create and deploy the contract. Then the authority will assign the ownership to the user who collected her biometrical information. Meanwhile, the user's transformed data is initialized through the variable $biohash$. Note that although the authority creates the smart contract, the owner of the contract is the user who can control the access policies. The $authorize$ function requires that the message sender (smart contract visitor) is the owner, then the owner can specify which verifiers, such as security check points, can obtain access to the contract for how long time.

The conditions to trigger $access$ function include: the message sender is in the whitelist, the corresponding specified access time is later than the current time, and the smart contract is still active. The $revoke$ function is designed for the authority to disable the smart contract when necessary. For instance, the data in the smart contract needs to be updated with new data. The $ObtainAuthority$ function is designed for authorized verifiers to check the authenticity of the creator (authority) of the smart contract.

B. User Registration by Authority

The whole identity management process starts from user registration by an authority, such as DMV. As shown in

Algorithm 1 $Smart\ Contract$

```
1: procedure ContractOperations:
2:     # Stores the address of this contract's owner
3:     address owner
4:     # Stores the hashed IPFS ID
5:     bytes biohash
6:     # Stores the addresses of all the permitted users
7:     mapping (address=>int) whitelist
8:     # Has the authority's address stored in the contract before deploying
9:     authority = contract address, e.g., "0x64a20b634734..."
10:    # If the contract is enabled
11:    bool enabled=true;
12:    # A constructor function which sets up the owner and load biometrical data to the smart contract
13:    constructor (user, hash){
14:        require(msg.sender==authority)
15:        owner=user
16:        biohash=hash
17:    }
18:    # An authorizing function which sets up who can access the smart contract for how long time
19:    authorize (verifier, time){
20:        require(msg.sender==owner)
21:        require(enabled==true)
22:        whitelist[verifier]=now+time;
23:    }
24:    # An retrieving function which sets up who can access the smart contract for how long time
25:    access (){
26:        require(whitelist[msg.sender]!=0)
27:        require(whitelist[msg.sender]>now)
28:        enabled==true
29:        return biohash
30:    }
31:    # A revoking function which disables the smart contract
32:    revoke (){
33:        require(msg.sender==authority)
34:        enabled==false
35:    }
36:    # A retrieving function which returns the creator of this smart contract
37:    ObtainAuthority (){
38:        require(whitelist[msg.sender]!=0)
39:        require(whitelist[msg.sender] > now)
40:        require(enabled==true)
41:        return authority
42:    }
43: end procedure
```

Algorithm 2, the authority first needs to collect the user's biometrical information along with other personal information, e.g., passport. We use fingerprint and passport ID to represent this kind of information. Then the collected fingerprint image is stored into IPFS distributed system with a return hash ID. The next step is to transform the hash ID and passport ID into a new hash value via a one-way hash function. Assume the smart contract template written by solidity programming language has been ready to use now, the authority will fill the required info to the template and submit it to the Ethereum blockchain system. Algorithm 3 uses the authority's Ethereum account to deploy the smart contract where the user account is specified as the owner who can authorize access permissions later. Note that the authority will not allow to store the collected image, the returned IPFS ID and other personal information after the user registration stage. The user has the full control privileges over her data. Also both Authority and the user need to have their own Ethereum user accounts beforehand.

Algorithm 2 Authority Operations

procedure $Register(Image, PassportID, ContractTemplate, AuthorityAccount, UserAccount)$:
    #Store a collected image into IPFS distributed system with a returned hash ID
    ID = StoreImage($Image$)
    #Transform the ID and other personal info, e.g., PassportID, to a new hash value
    $BioHash$ = TransformData(ID, $PassportID$)
    #Deploy the transformed data to smart contract
    $ContractAddress$ = Deploy($ContractTemplate$, $AuthorityAccount$, $UserAccount$, $BioHash$)
    Return $ContractAddress$
end procedure

Algorithm 3 Smart Contract Deployment

procedure $Deploy(ContractFile, AuthorityAccount, UserAccount, BioHash)$:
    # Open the smart contract file in a json format
    $data$=open($ContractFile$);
    # Extract the contract
    $contract$ = contract(data['abi'], data['bytecode'])
    # Deploy the contract with a user account and gas
    tx = contract($UserAccount$,$BioHash$)
    .transact($AuthorityAccount$, gas)
    $address$ = getTransactionReceipt(tx)
    return $address$
end procedure

C. User authorization

After the smart contract is created and deployed, a user can allow any verifier who has an Ehtereum account to access her ground-truth biometrical information in the blockchain platform. In order to offer the permissions, the user needs to know the smart contract's public address, the verifier's Ethereum account, the duration for that the verifier can access the smart contract, and the user's own Ethereum account. Algorithm 4 first checks if the provided smart contract address is valid, and then obtains the contract information, and proceeds to issue a new 'authorize' transaction to the existing contract with the aforementioned parameters. Note that this step will change the state of the existing smart contract and thus will incur a new transaction in the blockchain. This process requires the user to pay a certain amount of gas as the service fee. The returning value is a new transaction hash ID.

Algorithm 4 $Authorization$

procedure $authorize(address,verifier,time, account)$:
    # Validate smart contract address
    $ValidAddress$=CheckAddress($address$)
    # Obtain smart contract
    $contract$=getContract($validAddress$)
    # Issue transaction
    tx=Transact($contract$,'authorize'($verifier$,$time$), $account$, gas)
    $TransactionHash$=TransactionReceipt(tx)
    return $TransactionHash$
end procedure

D. Identity Verification

Algorithm 5 describes the steps to finish an identity verification. First of all, at the verifier's side, e.g., a security check point, a new fingerprint image needs to be collected on the scene and it will be compared with the one stored in IPFS by NIST Biometric Image Software (NBIS) system [10]. If the comparison result is positive, that indicates the user is the owner of the image stored in IPFS. Note that we use NBIS to check if two fingerprint images are from the same person in this implementation. The comparison module can be replaced by any other fingerprint identification technology. For example, machine learning techniques can replace NBIS for higher accuracy and flexibility.

Second, the verifier obtains the creator account information of the smart contract and confirms that it was the authority who deployed the contract. Note that the authority's smart contract user account information should be publicly available for any verifier to obtain and verify. Third, the verifier can retrieve the transformed biometrical information from the smart contract. The information is nothing but a hash value, through which, however, the verifier is unable to infer the original data. The verifier needs to use available data from the user to repeat the process of data transformation executed by the authority and converts them into a hash value. If the two values match, it indicates that the user on the scene is the same person who registered at the authority. Therefore, the verification passes. If any step above fails, the identity verification cannot go through.

Algorithm 5 Identity Verification

procedure $Verify(address,\ IpfsID,\ NewImage,\ VerifierAccount)$:
  $OriginalImage$=IPFS.getFile($IpfsID$)
  $match$=NBIScompare($OriginalImage$, $NewImage$)
  if $match$==true **then**
    $Authority$=ObtainAuthority($address$, $VerifierAccount$)
    if $Authority$==DMVaddress **then**
      $RealAuthority$=true
    end if
    $OldHash$=Access($address$, 'retrieve', $VerifierAccount$)
    $NewHash$=TransformData($IpfsID$, $PassportID$)
    if $OldHash$==$NewHash$ and $RealAuthority$=true **then**
      Verification succeeds
    end if
  end if
end procedure

Algorithm 6 Smart Contract Access

procedure $ObtainAuthority(Address,\ VerifierAccount)$:
  # Validate smart contract address
  ValidAddress=CheckAddress($address$)
  # Obtain smart contract
  $contract$=getContract($validAddress$)
  # Access smart contract for data
  $biohash$=Call($contract$,'ObtainAuthority', $VerifierAccount$)
  return $biohash$
end procedure

E. Evaluation Results

We used a fingerprint scanner to collect the fingerprint images for our experiments. Table I illustrates the average time each step takes. Overall, it takes less than one minute to finish the whole process. And the verification itself takes about 15 seconds without any optimization. Actually, except the verification process that needs to be done on the scene, other steps can be finished beforehand. Overall, the testing results are promising to streamline identity verification at critical public services, such as airport security check, without introducing a long time delay.

F. Security Analysis

There are a few scenarios that may lead our system to potential breaches. They are primarily located at the data entrance points, such as authority registration, user authorization and data retrieval from verifiers. For instance, if any of the systems is compromised, the corresponding private key may be stolen and abused for malicious purpose. We also assume that the verifier's end will always discard all collected biometrical information after the corresponding individual passes the security check, otherwise, the person's private information may be disclosed to the public. The public key management infrastructure is another possible point of weakness since we rely on it for authority and ownership verification. A decentralized solution may be applied to mitigate the risk, such as SCPKI [5].

TABLE I: Application test results

Process	Attribute	Time in sec
Registration	Fingerprint collection	∼3
Registration	IPFS uploading & hashing	∼2
Registration	Contract construction	∼10
Authorization	Add address to contract	∼10
Verification	Hash retrieval	∼10
Verification	Hash verifying	∼1
Verification	IPFS downloading	∼2
Verification	Fingerprint comparison	∼1

V. Related Work

There are many related works that use Blockchain technologies for identity management [11], [12], including Decentralized Identifiers (DIDs) [13], Self-Sovereign Identity, Veridium [14], etc. Some of the works also can realize self-sovereign feature. We introduce them as follows.

A. Decentralized Identifiers

Sovrin, Veridium, uPort and Veres One are using self-sovereign identities, which means the owners have the absolute and whole control to their registered data. A well-designed self-sovereign identity may enable the owner to visit, modify and revoke their data without permission from a third party. Sovrin, Veridium and Veres One take advantage of Decentralized Identifiers (DIDs) to implement self-sovereign identity systems. DIDs are provide a standard way for individuals and organizations to create permanent, unique, cryptographically verifiable identifiers under their owners' control. As a DID is generated randomly by an algorithm [15], and it totally belongs to the user instead of rentable to others, which is different from domain names, phone numbers and IP addresses.

A DID document has data required to interact with its DID. Typically a DID Document has three parts: proof purposes, verification methods, and service endpoints. Proof purposes, together with verification methods, is responsible to specify verification methods for given keys or protocols. Service endpoints are used to set and initialize an already trusted device. For example, in Veridium, the first part corresponds to the mechanisms used to authenticate such as keys, biometric templates and one share of a biometric data. The second part corresponds to a set of authorization information which points out what may modify the DID Document. The third part corresponds to a set of service endpoints that may be used to initiate interactions with devices.

B. Veridium and Horcrux Protocol

Proposed by Veridium, the Horcrux protocol is a method for securing biometric information registration and access [14]. The protocol is generalized for two or more biometric shares that can be stored across mobile devices and personal storage providers with redundancy for availability and safety. In Veridium, the biometric data's owner is able to authorize others get access to the data without permission from a third party. The owner can assert the identity transaction claim or authorize a verified and trusted third party to do so. The trust is diffused in its self-sovereign environment and is not controlled by any single or grouped organizations. Blockchain helps enable this environment by creating multiple department nodes like organizations and governments. As a result, department nodes mutually form distributed consensus and data are recorded in different places and hence, resistant to mistakes.

C. uPort

uPort allows the identity owners to control their personal identity and corresponding keys and data [16]. Owners can authorize others to access data, proceed digital services such as signing documents, control and send values on a blockchain, interact with smart contracts as well as applications, and encrypt data. The uPort enterprises can create identities for new customers and employees; establish a Know-Your-Customer process; build secure access-controlled environments with less friction, hold little sensitive information to reduce liability, maintain the network of vendors; establish an environment where identities have specific roles and nothing to do with actors.

D. Sovrin

In Sovrin, there are four principles for a developer to consider to generate a successful self-sovereign identity system [15]: (1) Governance: how to make others trust that this application is secure enough that the data is very hard to be stolen or accessed by others. (2) Performance: how can the self-sovereign identity be provided in a large scale. (3) Accessibility: how can the network ensure that identity is available to all. (4) Privacy: how can privacy be guaranteed in the network.

E. Comparison

One main difference between our work and the related works lies in that our implemented framework does not rely on the authentication over mobile devices. One key assumption for other works, such uPort, is that mobile devices are equipped with biometrical authentication capabilities and are always trustworthy, through which an individual can authorize blockchcain or other online transactions. However, our approach requires that the individual re-take his/her live biometrical information in the scene and compare it with the one that was registered by an authority, then connect the identification results to the ground-truth anchor in the smart contract. This approach ensures a higher security level and confidence of the identification outcomes for critical services, such as airport security check and border pass.

VI. Conclusion

We designed and built a new identity management framework that can integrate a user's transformed biometrical data to a smart contract through Ethereum blockchain platform. The framework enables data security via a distributed public ledger, privacy preserving via personal storage of sensitive biometrics, and self sovereign via flexible programming capabilities of smart contract. We used solidity programming language to implement the smart contract template and deployed it in the Ethereum test network. We also collected real fingerprint images for the experiments. The testing results demonstrate that the framework is promising to be used in practice. As for future work, we will implement more diverse access control policies and take advantage of the data model, format, and operations that are supported by Decentralized Identifiers (DIDs) to manage the identity framework.

VII. Knowledgement

This material is based upon work supported by the Center for Identification Technology Research and the National Science Foundation under #1650503.

References

[1] A. K. Jain, P. Flynn, and A. A. Ross, *Handbook of biometrics*. Springer Science & Business Media, 2007.

[2] "Yahoo! data breaches," https://en.wikipedia.org/wiki/Yahoo!_data_breaches.

[3] M. Hansen, P. Berlich, J. Camenisch, S. Clauß, A. Pfitzmann, and M. Waidner, "Privacy-enhancing identity management," *Information security technical report*, vol. 9, no. 1, pp. 35–44, 2004.

[4] F. Sabena, A. Dehghantanha, and A. P. Seddon, "A review of vulnerabilities in identity management using biometrics," in *2010 Second International Conference on Future Networks*. IEEE, 2010, pp. 42–49.

[5] M. Al-Bassam, "Scpki: a smart contract-based pki and identity system," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM, 2017, pp. 35–40.

[6] J. L. Wayman, "Biometrics in identity management systems," *IEEE Security & Privacy*, vol. 6, no. 2, pp. 30–37, 2008.

[7] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[8] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger."

[9] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[10] "NIST Biometric Image Software (NBIS)," https://www.nist.gov/services-resources/software/nist-biometric-image-software-nbis.

[11] O. Jacobovitz, "Blockchain for identity management," *The Lynne and William Frankel Center for Computer Science Department of Computer Science. Ben-Gurion University, Beer Sheva*, 2016.

[12] S. Muftic, "Blockchain identity management system based on public identities ledger," Apr. 25 2017, uS Patent 9,635,000.

[13] "Decentralized Identifiers (DIDs) v0.12," https://w3c-ccg.github.io/did-spec/.

[14] A. Othman and J. Callahan, "The horcrux protocol: A method for decentralized biometric-based self-sovereign identity," *arXiv preprint arXiv:1711.07127*, 2017.

[15] A. Tobin and D. Reed, "The inevitable rise of self-sovereign identity," *The Sovrin Foundation*, 2016.

[16] C. Lundkvist, R. Heck, J. Torstensson, Z. Mitton, and M. Sena, "Uport: A platform for self-sovereign identity," *URL: https://whitepaper. uport. me/uPort_ whitepaper_DRAFT20170221. pdf*, 2017.