CertOracle: Enabling Long-term Self-Sovereign Certificates with Blockchain Oracles

Shaoxi Zou, Fa Jin, Yongdong Wu*
Jinan University, Guangzhou, China
*Corresponding author

***Abstract*—Identity certificate is an endorsement of identity attributes from an authority issuer, and plays a critical role in many digital applications such as electronic banking. However, the existing certificate schemes have two weaknesses: (1) a certificate is valid only for a short period due to expiry of the issuer's private key, and (2) privacy leaks because all the attributes have to be disclosed in the attribute verification process. To overcome the weaknesses, this paper proposes a blockchain-based certificate scheme called CertOracle. Specifically, CertOracle allows a traditional certificate owner to encrypt the off-chain certificate attributes with fully homomorphic encryption algorithms. Then, the uploading protocol in CertOracle enables to post the encrypted off-chain attributes into the blockchain via a blockchain oracle in an authenticated way, i.e., the off-chain attributes and on-chain encrypted attributes are consistent. Finally, the attribute verification protocol in CertOracle enables anyone to verify any set of on-chain attributes under the control of the attribute owner. As the on-chain certificate attributes are immutable forever, a traditional short-term certificate is transformed into a long-term one. Besides, the owner of the on-chain certificate attributes can arbitrarily select his/her attributes to meet the requirements of target applications, i.e., the on-chain certificate has the self-sovereign merit. Moreover, the proposed scheme is implemented with fully homomorphic encryption and secure two-party computation algorithms, and its experiments show that it is viable in terms of computation time and communication overhead.**

***Keywords*—Digital certificate; blockchain oracle; fully homomorphic encryption; secure two-party computation**

I. Introduction

As an important way to identify persons or other objects, an identity certificate is a set of electronic credentials that are used to verify the certificate owner's identity. It can be an electronic ID card, an electronic graduation certificate, a health certificate, or other identity certificate issued by a Certificate Authority(CA) such as a government, university, hospital, etc. Since most digital identity schemes use digital certificates to uniquely identify people on the Internet, this will lead to the frequent use of digital certificates in daily lives.

Most existing digital certificate schemes are based on public-key cryptography. During the issuance of a certificate, the CA's private key will be used to sign on the attributes of the certificate. However, according to the specification of NIST, the private key of CA should be updated within three years [1], otherwise the security of certificate scheme may be compromised. Usually, the period of a private source-authentication key, such as signing key, is the same as the period of associated public key. Therefore, once the key of CA is refreshed, the verification on previously issued digital certificates (e.g., electronic ID card or electronic graduation certificate) may fail, and hence cause inconvenience or security vulnerability. Therefore, it's necessary to extend the life cycle of these digital certificates and verify identity attributes even if the CA's private key has expired.

Usually, digital identity certificates contain multiple user attributes, including user's private information. When users want to use certain services supported by verifiers, they have to present a part of attributes in their certificates. However, if the identity certificate is presented straightforwardly, all the attributes in the certificate will also be exposed. This is a coarse-grained representation that does not satisfy the principle of least leakage. Meanwhile, someone can't verify the certificate's authenticity when only some attributes of the certificate are displayed. Therefore, it is necessary to propose a method to perform fine-grained verification of attributes in certificate.

To solve the above problems of expiry of CA's private key and privacy leakage of user's attributes, Self-Sovereign Identity(SSI) [2] solutions are proposed to enable the owners to control their attributes with decentralized technologies [3] such as blockchain or distributed ledger. Presently, there are many mature decentralized identity schemes, such as Hyperledger *Indy* and *Civic* built on consortium blockchains and *ION* built on public blockchains. However, these proposed solutions mostly rely on on-chain issuers. It is difficult for them to tackle the off-chain certificates, i.e., the traditional digital identity certificates.

Since the blockchain is characterized by decentralization, immutability and openness, the on-chain certificate lifetime will be extended to long-term. Nonetheless, since the information on the blockchain is public and traceable, storing digital certificates on the blockchain in plaintext can lead to privacy leakage. Therefore, the attributes of on-chain certificates should be encrypted. To ensure the encrypted traditional certificates are uploaded into a blockchain authentically, blockchain oracle can be played as a middleware role in building a bridge between off-chain and on-chain.

This paper presents CertOracle, a management scheme for long-term and fine-grained certificates by transforming an off-chain traditional digital certificate into an on-chain certificate. Technically, CertOracle enables users to encrypt their digital certificates by themselves and ensures that the encrypted identity certificate is uploaded into the blockchain in an authentic way via a blockchain oracle. The on-chain encrypted attributes can be arbitrarily selected for verification under the control of the attribute owner. CertOracle ensures that on-chain certificate attributes are fine-grained verifiable and

long-term. Additionally, CertOracle is implemented with fully homomorphic encryption and secure two-party computation algorithms, and the experiments demonstrate that CertOracle is practical.

The remainder of this paper is organized as follows. Section II shows recent work. Section III introduces the preliminaries. Section IV describes the system model, security model, and the protocol details of CertOracle. Section V analyzes CertOracle. Section VI elaborates the CertOracle implementation and evaluates CertOracle with abundant experiments. Finally, Section VII draws conclusions.

II. Related Work

To feed blockchain with encrypted certificates, blockchain oracles need to verify the correctness of the encrypted certificate. According to recent research, the implementation of blockchain oracles could be divided into three categories [4]: TLS-based scheme, Enclave-based scheme, and Voting-based scheme.

TLS-based schemes are conducted to construct a protocol based on Transport Layers Security (TLS), such as TLS-N [5], DECO [6], etc. TLS is a protocol instrumented to guarantee message integrity and server authentication through the exchange of certificates during TLS-handshake. TLS-N, as a practical and decentralized blockchain oracle, enhances the audibility and reliability of web content based on the TLS. However, the scheme is less deployable, which requires some improvements to TLS. Therefore, DECO proposes a three-party protocol based on modifications of TLS client, which minimizes the modification requirements of the TLS protocol and enhances compatibility. CanDID [7] uses TLS-based oracles to fetch identity information securely from outside systems. However, the oracles are not viable for protecting the digital identity certificates issued by traditional identity systems.

Enclave-based oracle is constructed with Trusted Execution Environment (TEE), such as Intel Software Guard Extensions(Intel SGX) and TrustZone. Using the technology of Intel SGX, Town Crier [8] can encrypt and decrypt data requests from smart contracts and external data from data sources, securely managing sensitive information. But Town Crier highly depends on the trusted execution environment and may be vulnerable to side-channel attack [9].

Voting-based schemes, such as Augur, Oraclize [10], Chainlink [11] and Augur [12], motivate oracles to vote for data honestly through monetary rewards and penalties. Oraclize provides proof for the data it fetches, ensuring that the original data source is real and untampered. But its centralized model cannot guarantee that the service will always be available. ChainLink is a decentralized oracle solution on Ethereum. By querying multiple sources, ChainLink can avoid dependency of a single oracle. Augur is a prediction market platform oracle built on Ethereum. It allows users to vote on information from the outside world based on their tokens. Therefore, the prediction accuracy is limited by the scale of the platform, and the uneven distribution of tokens affects the credibility of results. Apart from that, voting-based schemes do not provide authenticity of the data and are not suitable when data is not publicly available, especially digital certificates.

III. Preliminaries

A. Digital Certificate

Digital certificate such as X.509 [13] is an electronic document issued by CA. Since the user's digital identity certificate is issued by a trusted center, it can be used to prove personal attributes such as age, gender, and country. To produce a certificate, the attributes will be compressed using a one-way hash algorithm such as 256-bit Secure Hash Algorithms (SHA256) [14] first. Then a CA adopts a digital signature algorithm such as ECDSA and its private key to sign on the hash value. As a result, anyone can verify the signature with the CA's public key.

Taking the hash algorithm SHA256 and digital signature algorithm RSA as examples, CA chooses randomly two large prime numbers p and q, and computes $N = pq$ and $\phi = (p-1)(q-1)$ first. Secondly, it chooses a random integer $e(1 < e < \phi)$, so that $gcd(e, \phi) = 1$, and computes $d(1 < d < \phi)$, so that $ed \equiv 1 \mod \phi$. Then CA owns public key (N, e) and private key d. Thirdly, to generate hash value of certificate content M with l bits, CA adds a bit "1" at the end of the message, then add k "0", and finally, fills in 64 bits representing the length of the message. k is the smallest non-negative integer that satisfies:

$$l + 1 + k + 64 = 0 \mod 512$$

The padded message is divided into several message blocks $M_0, M_1, \cdots, M_n$ in unit of 512 bits, and the M_i is encoded using big-endian and represented as $W_i^0, W_i^1, \cdots, W_i^{15}$. Then CA construct W_i tables for every block according to $W_i^0, W_i^1, \cdots, W_i^{15}$. Then, for $t = 16 \ldots 64$,

$$W^t = \sigma_1(W^{t-2}) + W^{t-7} + \sigma_0(W^{t-15}) + W^{t-16}$$

After initializing $A \ldots H$, CA runs 64 rounds of compression function with W_i table for every block, denoted as $A_{64} \ldots H_{64} = Com(A \ldots H, W_i)$, where $Com(\cdot)$ is the hash operation of one block. The $A_{64} \ldots H_{64}$ will be assigned to $A \ldots H$ for compressing the next message block. After performing $Com(\cdot)$ on every block sequentially, CA gets the hash value h.

Finally, CA computes $\sigma = h^d \mod N$ (Hereafter, we omit the padding processing in the digital signature algorithm for the sake of simplicity) and obtains σ as signature. To verify the signature, any entity can compute $h = \sigma^e \mod N$ and calculate h' using SHA256 with certificate to be verified. If the value of $h = h'$, it means that the certificate has not been tampered with and is indeed issued by the CA.

B. Secure Multi-Party Computation

Secure multi-party computation (SMPC) protocol [15], is a cryptographic primitive that allows several participants to jointly calculate some functions and outputs correct results without revealing the original input data of the participants. It is suitable for distributed networks like blockchain because it deals with security and trust issues in a distributed environment.

As the simplest one in SMPC, a two-party computation (2PC) ensures that two non-trusted parties attempt to securely evaluate a function. Technically, it specifies a random

process that maps pairs of inputs to pairs of outputs, i.e., $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where each party has one private input and one output. For example, Goldreich-Micali-Wigderson (GMW) [16] protocol is a secret-shared-based 2PC scheme and can evaluate NOT, XOR and AND. For instance, for an XOR gate with input values $x = \{x_i\}$ and $y = \{y_i\}$ and output value $z = \{z_i\}$, two parties can compute their shares of the output value as $z_i = x_i \oplus y_i$ locally, where $\oplus$ is eXclusive OR (XOR). The evaluation of AND gate can be achieved using multiplication triples [17]. Multiplication triples are random shares $\{a_i, b_i, c_i\}$, $i \in \{1,2\}$ with $(c_1 \oplus c_2) = (a_1 \oplus a_2) \wedge (b_1 \oplus b_2)$. They can be generated using 1-out-of-4 oblivious transfer protocol in the offline set-up phase which precedes the online phase. During online phase, the parties can use these multiplication triples to mask the input shares of the AND gate. Both parties exchange $u_i = x_i \oplus a_i$ and $v_i = y_i \oplus b_i$, and compute $u = u_1 \oplus u_2$ and $v = v_1 \oplus v_2$. Then the output can be computed as

$$z_1 = (u \wedge v) \oplus (b_1 \wedge u) \oplus (a_1 \wedge v) \oplus c_1 \tag{1}$$

and

$$z_2 = (b_2 \wedge u) \oplus (a_2 \wedge v) \oplus c_2 \tag{2}$$

for each party from their private input x and y.

C. Fully Homomorphic Encryption

A Fully homomorphic encryption(FHE) scheme allows homomorphic evaluation of ciphertext addition and multiplication an arbitrary amount of times without decrypting the ciphertext or revealing the secret key [18]. That is to say, for any messages x and y, and a constant α,

$$\mathrm{FHE}(\alpha x + y) = \mathrm{FHE}(\alpha) \cdot \mathrm{FHE}(x) + \mathrm{FHE}(y) \tag{3}$$

for a fully homomorphic encryption algorithm $\mathrm{FHE}(\cdot)$.

TFHE(Torus-FHE) [20] is the third generation of FHE based on the hardness of Learning with Errors(LWE) problem and its ring variant(Ring LWE). Running over the real Torus $\mathbb{T} = \mathbb{R} \mod 1$, TFHE can fix the message space as a discrete subset $\mathcal{M} \subseteq \mathbb{T}$. A message $\mu \in \mathcal{M}$ can be encrypted by computing

$$b = s \cdot a + \mu + e \tag{4}$$

where $s \in \{0,1\}^n$ is the secret key, $a \in \mathbb{T}^n$ is uniformly random and e is sampled from an error distribution over $\mathbb{T}$. Then, the ciphertext is a pair $c = (b, a) \in \mathbb{T}^{n+1}$. In order to decrypt, phase φ_s is introduced to compute

$$\varphi_s(c) = b - a \cdot s \tag{5}$$

After rounding $\varphi_s(c)$ to the nearest point in $\mathcal{M}$, message μ can be obtained.

TFHE can transform an arbitrary function into a boolean circuit, then uses bit-wise homomorphic evaluation on ciphertext c_1, c_2 as each gate's input. For example, the AND gate in TFHE is done by evaluating $c = \mathsf{BootsAND}(c_1, c_2, bk)$, in which bk is a key used for bootstrapping. Other operations are also supported with similar gate operations, such as NOT, NAND, OR, XOR, etc. Since TFHE can evaluate arbitrary boolean circuits, it is very suitable for functions that can be represented by several logical operations.

IV. The Present Scheme CertOracle

CertOracle aims to feed the blockchain-based system with encrypted identify attributes so as to provide long-term self-sovereign identity. That is to say, the on-chain encrypted attributes can be used for fine-grained verification, ensuring that the attribute holder can have complete autonomy over their personal identity attributes for a long time. Thus, the scheme shall achieve the following goals:

- **Privacy**: The user's attributes in certificate are protected during the on-chaining process and on the chain.
- **Unforgeability**: The user's encrypted on-chain certificate attributes are verifiable as the traditional certificates and cannot be forged when uploading.
- **Fine-grained verification**: The on-chain attributes of certificates can be verified in a fine-grained way securely.

The parameters of CertOracle are shown in Table I.

TABLE I. Notations

Variable		Definition
$Attr_i$	:	The i-th attribute
V_i	:	The i-th attribute set V
h_i	:	The i-th bit of hash value h
X'	:	The FHE encryption of a message X
$\hat{X}$	:	The obfuscated value of a message X
$\{X_i\}$	:	Vector $X = \{X_0, X_1, \ldots, X_{\lvert X\rvert-1}\}$
$cert$	:	Identity certificate
σ	:	Signature of a certificate

A. System Model

As shown in Fig. 1, the blockchain-based identity system supports the management and control of users' identities. When a user has a digital identity certificate generated by an off-chain CA, he/she can upload the digital identity certificate to the system through blockchain oracle of CertOracle. To ensure users' self-sovereign of the on-chain certificates, CertOracle permits users to encrypt private information of digital certificates by themselves. Technically, oracle verifies the encrypted certificate, in order to confirm that the on-chain certificate is produced authentically. Then any verifier can check the selected attributes in a trusted and privacy-preserving way.

With reference to Fig. 1, CertOracle includes four components as follows.

- User: A user owns a certificate which can be represented as $cert = (ID, \mathcal{M}, \sigma)$, where ID is an identifier of the user, $\mathcal{M}$ is the attributes of certificate and σ is signature generated by CA. $\mathcal{M}$ can be seen as an array of $\{Attr_i, V_i\}$, where V_i is the value of the attribute $Attr_i$, and signature $\sigma = sign(hash(\mathcal{M}))$ for attributes in the certificate with hash algorithm $hash(\cdot)$ and signing algorithm $sign(\cdot)$. The user has a pair of self-generated FHE keys (pk_U, sk_U), and can cooperate with the oracle to prove his encrypted attributes in the certificate.

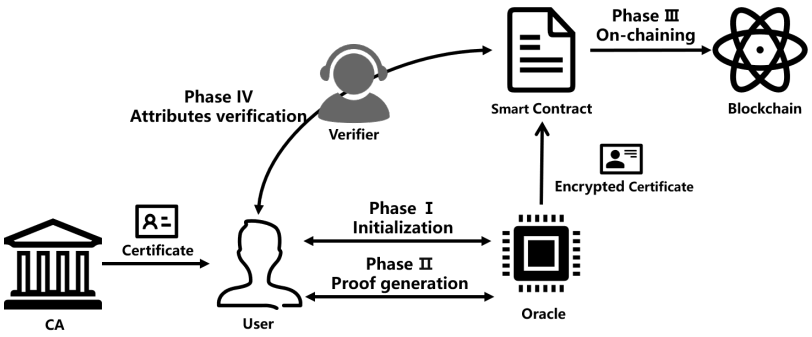

Fig. 1. The System Model of CertOracle.

- Oracle: Oracle has the trusted public keys of the CAs so as to verify the signatures of the certificates. In addition, by cooperating with users, oracle will prove/disapprove the encrypted certificate submitted from the user.
- Smart Contract: It can upload the encrypted attributes of certificates after attributes verification. Meanwhile, It can fetch any set of on-chain attributes and provide them to the verifier to check user's claims.
- Verifier: When a user likes to prove his attributes to a verifier, a verifier can validate on-chain encrypted attributes by interacting with the user and smart contract.

B. Security Model

With reference to Fig. 1, there are 5 components or participants. CA is assumed to be honest and trusted by all. However, CA's private key may be updated irregularly, resulting in a decrease in the credibility of the old digital certificate. Therefore, only when the CA's private key is valid, the digital certificate issued by the CA is credible, otherwise the verification of the previously issued digital certificate will be considered to be unreliable. The security assumptions of the other components are as follows:

- Oracle is semi-honest. It honestly follows the protocol of CertOracle and will not be corrupted by users, but is curious about the personal attributes of the certificate.
- User is untrustworthy. A malicious user wants to 1) upload a forged digital certificate to blockchain through oracle; 2) deceive the verifier with fake attributes in order to obtain unauthorized services. In addition, the ciphertexts sent to the user are assumed to be well-formed. This assumption can be met with padding such as OAEP (Optimal Asymmetric Encryption Padding).
- Smart contract, as a part of the blockchain system, is trusted and can provide reliable services.
- Verifier is semi-honest. He follows the protocol to verify some attributes of the users but attempts to obtain user's attributes that do not need to be verified

C. CertOracle Protocol

In Fig. 1, CertOracle comprises of four phases. 1) *Initialization* (Phase I) configures the parameters in CertOracle; 2) *Proof generation* (Phase II) validates the authenticity of the encrypted certificate; 3) *On-chaining* (Phase III) uploads the verified attributes in the certificate; 4) *Attribute verification* (Phase IV) checks the on-chain encryption of the user attributes. In this protocol, assume that the size of hash value is $(n_1 + 1)$. Without loss of generality, assume every attribute is binary, and the number of attributes is $(n_2 + 1)$.

1) Phase I (Initialization): In the initialization stage, a user firstly sends a request to establish a connection with the oracle, and runs FHE key generation function to get public/private key pair (pk_U, sk_U). The user keeps the private key sk_U during the proof generation and verification process for the encrypted attributes, and publishes the public key pk_U to oracle. Then the user encrypts some private attributes $\{Attr_i, V_i\}$ in $cert$ bit by bit using pk_U, and publishes other attributes of certificate, such as Version, Certificate Serial Number, Algorithm Identifier, and Validity Period in Certificate Information in the certificate format like X.509. In this paper, the public attributes are omitted, unless otherwise stated. Denote the ciphertexts $V'_0, V'_1, \ldots, V'_{n_2}$. Denote the encrypted certificate $cert' = (ID, \{Attr_0, V'_0\}, \ldots, \{Attr_{n_2}, V'_{n_2}\}, \sigma)$. The user constructs $ProofRequest$ as

$$(op, ID, \{Attr_0, V'_0\}, \ldots, \{Attr_{n_2}, V'_{n_2}\}, pk_U, \sigma)$$

and sends $ProofRequest$ to the oracle. In $ProofRequest$, op is the operations on each attribute $\{Attr_j, V'_j\}$, such as `Insert`, `Update` and `Delete` operation on attributes.

After receiving $ProofRequest$, the oracle will verify the signature σ against the encryption of attributes. To this end, the oracle calculates the hash value of the ciphertext and runs the signature verification function with the public key of CA. If the verification result is positive and the certificate is not on the blockchain, the oracle starts to run the following phases.

2) Phase II (Proof Generation): For any digital certificate generated from the hash-then-sign algorithm, the proof process of CertOracle can be divided into two solutions. If the user cannot keep connected with the oracle during proof generation, non-interactive proof protocol can be used; otherwise, interactive proof protocol is recommended

Non-interactive proof protocol: As shown in Fig. 2, if user can not stay online during proof generation, the ora-

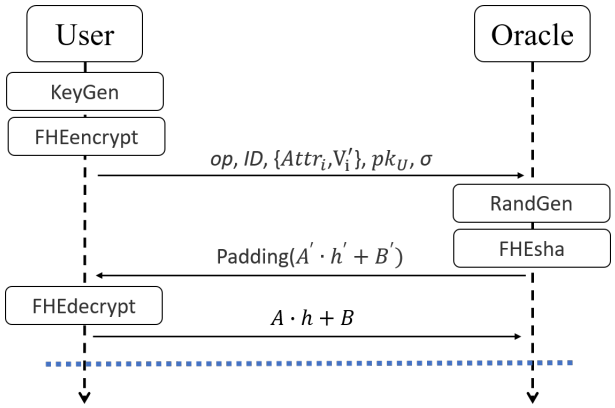

Fig. 2. The Non-Interactive Proof Protocol in CertOracle.

cle utilizes properties of fully homomorphic encryption to realize the proof process. Specifically, the oracle holds the $cert' = (ID, \{Attr_i, V_0'\}, \ldots, \{Attr_{n_2}, V_{n_2}'\}, \sigma)$. It uses pk_U to encrypt each attribute $\{Attr_i, V_i\}$ bit by bit.

To calculate the hash value of the fully homomorphic encryption of the certificate attributes, the oracle will run algorithm FHEsha whose input is the encryption of messages and whose output is the fully homomorphic encryption of hash of the messages[1]. That is to say, the oracle gets the encrypted hash

$$h' = \{h_0', h_1', \ldots, h_{n_1}'\} = \text{FHEsha}(V_1', V_2', \ldots, V_{n_2}') \quad (6)$$

where V_i' is an FHE encryption of the i^{th}-bit message V_i, and h_i' is an FHE encryption of the i^{th}-bit of the hash value $h = hash(V_0, V_1, \ldots, V_{n_2})$.

Moreover, the oracle generates two random arrays $A = \{a_0, a_1, \ldots, a_{n_1}\}$, and $B = \{b_0, b_1, \ldots, b_{n_1}\}$, and encrypts A and B to $A' = \{a_0', a_1', \ldots, a_{n_1}'\}$ and $B' = \{b_0', b_1', \ldots, b_{n_1}'\}$ using user's public key pk_U respectively, for all $i = 0, 1, \ldots, n_1$,

$$a_i' = \text{FHEencrypt}(a_i) \quad \text{and} \quad b_i' = \text{FHEencrypt}(b_i) \quad (7)$$

and obfuscates h' as [2]

$$\hat{h}_i' = a_i' \cdot h_i' + b_i'. \quad (8)$$

Oracle can pack[3] $\{\hat{h}_i'\}$ into $\hat{h}'$ and pad the ciphertext by[4]

$$BLOB = \text{FHEencrypt}(2^k) \cdot \hat{h}' + \hat{h}', \quad (9)$$

in which k is the number of bits of $\hat{h}'$, and obtains a blob equivalent to $\hat{h}'||\hat{h}'$.

[1]A hash function of plaintext can be translated into the encryption of the hash function of FHE ciphertext with google transpiler. Please refer to *https://research.google/pubs/pub50428/*

[2]$\{h_i'\}$ can be obfuscated in batch for quick process in Eq.8. Similar for $\{V_i'\}$ in Eq.12 and $\{\Delta_i'\}$ in Eq.15.

[3]Multiple LWE ciphertexts can be packeted into a ciphertext.

[4]Assume that the value of $\hat{h}$ is less than half of the FHE domain. Similar for $\hat{V}$ in Eq.13 and $\hat{\Delta}$ in Eq.16

Afterwards, the oracle sends $BLOB$ to the user for decrypting. The user uses his sk_U to decrypt the $BLOB$ by

$$blob = \text{FHEdecrypt}(BLOB) \quad (10)$$

and determines that the $blob$ conformed to the format of $\hat{h}||\hat{h}$. If the format is correct, the user returns $\hat{h}$. The oracle can recover

$$h_i = (\hat{h}_i - b_i)/a_i \quad (11)$$

and the hash value $h = \{h_0, h_1, \ldots, h_{n_1}\}$ for proving the authenticity of the encryption $\{V_i'\}$ of the certificate attributes.

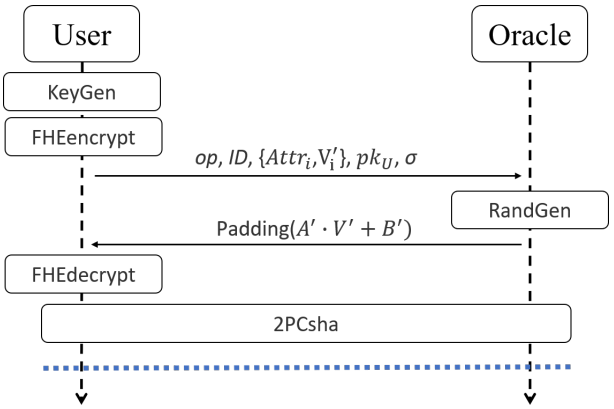

Fig. 3. The Interactive Proof in CertOracle.

Interactive proof protocol: as shown in Fig. 3, the oracle will interactive with the user using secure two-party computation based on secret sharing. The oracle holds the $cert' = (ID, \{Attr_0, V_0'\}, \ldots, \{Attr_{n_2}, V_{n_2}'\}, \sigma)$. Similar to the non-interactive solution, the oracle generates two random arrays $A = \{a_0, a_1, \ldots, a_{n_2}\}, B = \{b_0, b_1, \ldots, b_{n_2}\}$, and encrypts them to $A' = \{a_0', a_1', \ldots, a_{n_2}'\}, B' = \{b_0', b_1', \ldots, b_{n_2}'\}$ using pk_U as Eq.7. The oracle obfuscates the encrypted attribute V' as

$$\hat{V}_i' = a_i' \cdot V_i' + b_i' \quad (12)$$

After obfuscating, the oracle can pack $\{\hat{V}_i'\}$ into $\hat{V}'$ and send the padded ciphertext

$$BLOB = \text{FHEencrypt}(2^k) \cdot \hat{V}' + \hat{V}', \quad (13)$$

in which k is the number of bits of $\hat{V}'$, to the user. The user uses sk_U to decrypt $BLOB$ according to Eq.10 and determines that the $blob$ is equivalent to $\hat{V}||\hat{V}$, that is, equivalent to $\{\hat{V}_i\}||\{\hat{V}_i\}$.

To calculate the digest value of the encrypted certificate, the oracle and user jointly run 2PCsha based on boolean sharing. According to Subsection III-B, the user provides $\{\hat{V}_i\}$ and the oracle provides $A = \{a_0, a_1, \ldots, a_{n_2}\}$ and $B = \{b_0, b_1, \ldots, b_{n_2}\}$ to complete the 2PCsha protocol.

$$\{h_0, \ldots, h_{n_1}\} = \text{2PCsha}\left(\frac{\hat{V}_0 - b_0}{a_0}, \ldots, \frac{\hat{V}_{n_2} - b_{n_2}}{a_{n_2}}\right) \quad (14)$$

In the 2PCsha scheme, the deobfuscation circuit will securely produce $V_i = \hat{V}_i - b_i)/a_i$ in the internal wires of the boolean circuit, and then perform the hash circuit to output the hash value h for attribute proof.

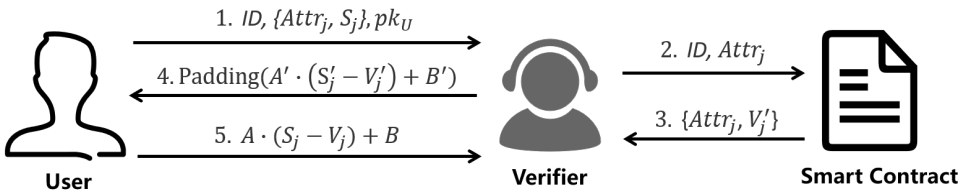

Fig. 4. The Workflow of Attribute Verification.

3) Phase III (On-Chaining): After recovering the hash value h, the oracle can check the authenticity of the certificate signature with the signature verification algorithm. If and only if the recovered hash h matches the signature, the oracle determines that $\{Attr_0, V_0'\}, \{Attr_1, V_1'\}, \ldots, \{Attr_{n_2}, V_{n_2}'\}$ in $ProofRequest$ are authentic and then uploads the encrypted certificate to the blockchain.

After oracle generates the proof of the encrypted certificate, smart contract checks if the user has certificate attributes on the blockchain. If not, it creates a DID (Distributed IDentity) for the user. If the user already exists, the smart contract verifies whether the user has permission to perform the specified operation through *op* such as Insert, updates or deletes the corresponding attributes in the blockchain.

4) Phase IV (Attribute Verification): As the certificate attributes are encrypted and stored in the blockchain, anyone can query and verify user's attributes publicly without compromising the privacy of the attribute owner. Furthermore, in order to meet the requirement of self-sovereign identity, querying and verifying need the assistance of the owner as shown in Fig. 4.

Technically, when the verifier needs to check whether the value of the attribute $Attr_j$ is S_j, $j = 0, 1, \ldots, n_3$, $n_3 \leq n_2$, it will encrypt S_j to S_j' using the user's public key pk_U. After that, the verifier generates two random arrays $A = \{a_0, a_1, \ldots, a_{n_3}\}$ and $B = \{b_0, b_1, \ldots, b_{n_3}\}$, and retrieves the encryption V_j' of the attribute $Attr_j$ from the blockchain. To securely authenticate the user, the verifier computes

$$\hat{\Delta}_i' = \Delta_i' \cdot a_i' + b_i' \tag{15}$$

where a_i', b_i' are ciphertext of a_i, b_i according to Eq.7 and Δ_i' represents i^{th}-bit of $S_j' - V_j'$. Then the verifier packs $\{\hat{\Delta}_i'\}$ into $\hat{\Delta}'$ and sends padded blob

$$BLOB = \text{FHEencrypt}(2^k) \cdot \hat{\Delta}' + \hat{\Delta}', \tag{16}$$

in which k is the number of bits of $\hat{\Delta}'$, to the user.

After the user decrypts $BLOB$ with private key sk_U, confirms that the decryption result conforms to the structure of $\hat{\Delta}||\hat{\Delta}$, and returns the obfuscated plaintext $\hat{\Delta}$ to the verifier, the verifier deobfuscates $\hat{\Delta}_i$ as

$$\Delta_i = (\hat{\Delta}_i - b_i)/a_i. \tag{17}$$

The verifier confirms the selected attributes of the user if and only if all the Δ_i are 0 (The decision policies may depend on the applications).

V. Analysis of CertOracle

As trust in identity certificates is transferred to the blockchain, the life cycle of these certificates can be extended to be the same as the lifetime of the blockchain. So these certificates become long-term certificates. This section discusses how CertOracle meets the goals to enable long-term self-sovereign certificates.

A. Privacy

In our protocol, the attributes of the certificate are encrypted bit by bit using TFHE. The attributes of on-chain certificates are protected to guarantee user's self-sovereign over his identity. When CertOracle is running, oracle is semi-honest. It wants to know the plaintext of encrypted attributes in $ProofRequest$. Since oracle just gets pk_U, it cannot learn the plaintext V_i from $\{Attr_i, V_i'\}$. Meanwhile, oracle can't know the plaintext of $V_1', V_2', \ldots, V_{n_2}'$ through $hash$ because of the non-reversibility of hash function.

In the non-interactive solution, the security is based on TFHE. There are many bit-wise operations in hash function. With the public key pk_U, the oracle of CertOracle can encrypt the attributes of certificate bit by bit so that it can calculate the hash value of the ciphertext message by itself. In the interactive solution, the security of the scheme is based on the GMW protocol. During 2PCsha, $\hat{V}_i$, A, and B are shared between oracle and user using boolean sharing. Thus, they look like uniformly distributed random data and this prevents the leakage of the data. A semi-honest adversary corrupting at most one of the participants can just observe secret shares of other's inputs. Meanwhile, the oracle can't learn the internal states of the circuit.

Moreover, we pad the output of Eq. 8, Eq. 12 and Eq. 15, that is, repeat the plaintext before encrypting it. By letting the decryptor check whether the decryption result meets a specific format, our system is resistant to potential fault injection attacks [19] from oracles and verifier.

B. Unforgeability

Before users encrypt their certificates, malicious users may forge attributes by modifying the critical payload $\{Attr_i, V_i'\}$ into another attribute's ciphertext $\{Attr_i, \tilde{V}_i'\}$. However, because of the collision resistance of hash function, it is impossible to find a $\tilde{V}_i'$ replacing the true data V_i' to get the same hash. In the non-interactive solution, malicious users may submit forged $ProofRequest$ but return true h when oracle sends h' back to decrypt after FHEsha. This may result in fake attributes being successfully verified. However, CertOracle obfuscates the h' by multiplying and adding a random number. It is

hard for malicious users to find out the relationship between obfuscated data and real data, and then forge data to pass verification. In the interactive solution, malicious users may submit forged $Proof Request$ but provide true data in the process 2PCsha. But they can't achieve because of obfuscation. In conclusion, CertOracle can effectively prevent forged certificates from being uploaded.

C. Fine-Grained Verification

User can upload multiple certificates to the blockchain and aggregate the attributes of certificates. When a user needs to use some attributes from different certificates to meet the requirement of a target application, he can construct a claim about the attributes based on the on-chain certificate. The on-chain attributes are encrypted and the verification process requires user's assistance which prevents all attributes from being exposed. Meanwhile, CertOracle ensures that the user attribute can be verified in a fine-grained manner, thereby ensuring the principle of least leakage. In addition, when a verifier checks the users' attributes $\{Attr_i, V_i'\}$, he obfuscates the attributes to be verified. Since the malicious user does not know the random array A, B, they cannot forge $\hat{V}$ to pass verification. However, verifiers may utilize deobfuscation to learn attributes that do not need to be verified. This can be mitigated by limiting the number of interactions between the verifier and the same user.

VI. Implementation

The scheme is implemented and tested with an AMD Ryzen 7 4800U CPU with 1.8 GHz and 4 GB RAM, running Ubuntu 20.04 LTS. The implementation adopts TFHE library and ABY [21]. Although the scheme is suitable for all secure hash algorithms, SHA256 is used as an example in the implementation. In the experiments, the attribute value is an integer, and one user may have certificates from different CAs. As shown in Fig. 5, when a user clicks "Upload Certifiacte" button, the certificate will be uploaded to the chain via CertOracle. When a user clicks "Create Claim" button, an attribute verification process will be activated and the off-chain certificates can be uploaded through Oracle and shown in *CertificateList*. Thus, the user can select an on-chain certificate and create a claim according to the service's requirement.

A. TFHE Setting

In CertOracle, we encrypt the certificate bit by bit using TFHE library. In the TFHE library, it needs at least one FFT processor to run the library. We choose FFTW3, which claims to be the fastest free FFT processor available, as the component. Meanwhile, the library provides two types of keys: cloud key and secret key. The cloud key is essentially a bootstrapping key used for cloud to perform homomorphic operations as well as encrypt constants, whereas the secret key is for encrypting the initial message and decrypting. Although addition and multiplication are not directly provided in TFHE library, we can utilize logical operations to build addition and multiplication of arbitrary bits.

In the TFHE library, the parameters are only implemented for 80-bit and 128-bit of security. Both these parameters are estimated using LWE estimator so that they can resist known attacks integrated in LWE estimator, such as primal attack and dual-lattice attack. As cryptographers recommend at least 128-bit security in practice [22], we use the 128-bit security version of parameter set in the implementation. However, the TFHE library does not yet support ciphertext packing. Therefore we ignore the realization of padding in the experiment. Instead, the user directly decrypt the output of Eq. 8, Eq. 12 and Eq. 15 using $\mathrm{FHEdecrypt}$.

B. Basic Bitwise Operators

In the SHA256, there are many additional operations in the algorithm. The efficiency of CertOracle will be mainly limited by three basic operators: adder, multiplier and obfuscator.

1) Bitwise Addition: Addition between two integers can be implemented using two XOR gates, a AND gate and a MUX gate. The adder is based on the simple logic equations

$$S = X \oplus Y \oplus V_{in}' \tag{18}$$

and

$$V_{out}' = MUX_{(X \oplus Y)}(V_{in}', X \cdot Y) \tag{19}$$

where X and Y are binary, V_{in}' is a bit that is carried in from the one less significant bit, and V_{out}' is a bit that is to be carried to the next significant digit. In $MUX_{X \oplus Y}(V_{in}', X \cdot Y)$, the gate outputs V_{in}' if $X \oplus Y == 1$ and outputs $X \cdot Y$ otherwise. As the addition in our protocol is 16-bit addition or 32-bit addition, a 16-bit addition costs about 1.33s.

2) Bitwise Multiplication: It can be implemented using a combination of adder and multiplexer. The multiplexer, worked as $MUX_{B_{[i]}}(0, A)$ with A and B are 16-bit, outputs 0 if i-th bit of B is equal to 1 and outputs A otherwise. With the multiplexer worked on 16-bit, it will run 16 times and then perform addition every time. In short, the 16-bit multiplier includes 16 multiplexers and $16 \cdot 16$ adders and takes about 11.457s on average which is expensive in time consumption.

3) Bitwise Obfuscation: With the bitwise adder and multiplier, we can evaluate the process of obfuscation. In both proof solutions, we need to obfuscate the data to be decrypted by users to prevent users from cheating. According to the number of obfuscated data, the oracle generates A, B. After the oracle selects the bytes to obfuscate, the data multiply and add different random numbers to get the same amount of obfuscated data. When deobfuscating, the oracle performs one subtraction and division to recover the original data. In the experiments, the obfuscation protocol takes about 127.939s to randomly obfuscate 10 bytes.

C. Non-Interactive Proof Protocol

The non-interactive solution relies on TFHE library using parameters with security level of 128-bit. Since the additions modulo 2^{32} in SHA256 need many gates, it is time-consuming for the oracle to perform a complete SHA256. To improve the efficiency of the protocol, a user may perform some steps of SHA256, including message padding, dividing, and W table construction. The user creates the W table for each block, and then sends the W tables to the oracle. The oracle only needs to perform the compression function.

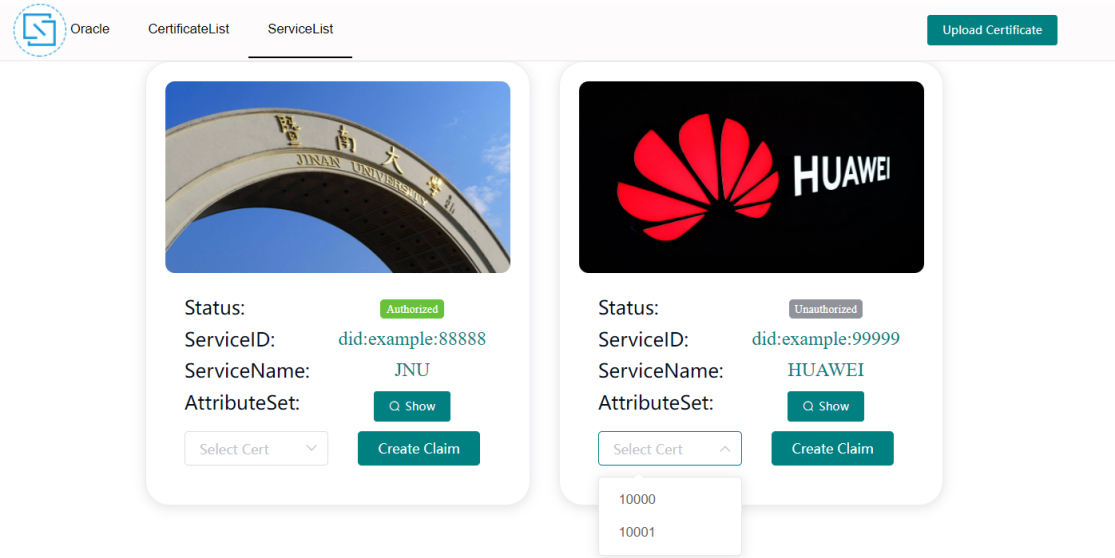

Fig. 5. The User-Interface of CertOracle.

The protocol is evaluated from two aspects. On the user side, we evaluate the function of key generation, procedure of partial SHA256(Message padding, Message dividing, and W table construction), encryption of 32 bits, and decryption of 16 bits. First, cloud key and secret key are generated in KeyGen. Secondly, PartialSHA256(64bytes) is to perform partial SHA256 calculation on 64bytes message and generate two blocks message. Thirdly, the time of certificate encryption is the time of multiple Enc(32bits) and the Dec(16bits) is a part of FHEdecrypt. On the oracle side, the result of running FHEsha with 64bytes and 128bytes messages are investigated. For instance, FHEsha256(64Bytes) is a homomorphic compression function for 64 bytes messages.

In the Table II for the evaluation results, the computation time of FHEsha in Table III doesn't include the time of obfuscation. Meanwhile, since the time spent in FHEsha is relatively large, the time on the user side is almost negligible. In all, the total time of the solution is the sum of FHEsha time plus obfuscation time.

TABLE II. EVALUATION OF NON-INTERACTIVE PROOF

Procedure	Time	Size
KeyGen	682.1ms	108.4MB / 108.4MB
PartialSHA256(64bytes)	0.041ms	2048bits
Enc(32bits)	0.42ms	0.077MB
Dec(16bits)	0.06ms	16bits
FHEsha(64bytes)	1834s	4.953MB
FHEsha(128bytes)	3668s	4.953MB

D. Interactive Proof Protocol

ABY is a framework for efficient mixed-protocol secure two-party computation in the interactive solution. It combines secure computation schemes based on arithmetic sharing, boolean sharing, and Yao sharing. The boolean sharings are used to evaluate functions represented as boolean circuits using the GMW protocol. Though ABY allows developers to manually specify which part of a function should be computed in which sharing to achieve the best overall efficiency, we use the boolean sharing part of ABY to test the present protocol. This framework, using the semi-honest adversary model, works like a virtual machine that abstracts secure computation protocols. Thus, the implementation of CertOracle constructs 2PCsha using GMW protocol [16] in the ABY.

Using an XOR-based secret sharing scheme to share a variable, the boolean sharing can perform the evaluation on NOT, XOR, AND, and others. While XOR gate and NOT gate can be evaluated locally, AND gate is evaluated using a precomputed boolean multiplication triple which reduces the impact of the latency during the online time. Next, we introduce the evaluation of the interactive solution in a local area network(LAN) environment.

In the ABY framework, the OT (Oblivious Transfer) extension protocol uses few base-OTs to quickly compute a large number of OTs so as to generate multiplication triples. As the base-OTs are performed once when the connection between the client and the server is established, their computation time is omitted in the total time. As a result, the average run-time is 377.595s and the communication overhead is 49,964 bytes.

The main part of 2PCsha comprises setup phase and online phase. The setup phase includes the pre-computation of some nonces generation for one-time pad operation and the multiplication triples for the AND gates. The run-time and communication overhead depend on the AND depth of the circuit, that is the number of messages. The online phase takes place after the setup phase is done and the inputs to the circuit are supplied by both parties. During online phase, two independent 2-bit messages have to be transmitted per layer of AND gates. This interaction in online phase causes a performance bottleneck, especially in high latency networks. Meanwhile, the ABY framework implements load balancing for the setup phase. As a consequence, the computation and communication cost are equally distributed among the two parties.

TABLE III. RUN-TIME AND COMMUNICATION OVERHEAD OF INTERACTIVE PROOF

Message Size	Time[ms]			Communication[bytes]		
	Setup phase	Online phase	Total	Setup phase	Online phase	Total
64	262.579	3,507.48	3,770.07	6,607,027	113,993	6,721,020
128	380.033	4,991.22	5,371.27	10,774,766	185,713	10,960,479
196	435.822	6,505.23	6,941.07	14,948,640	257,417	15,206,057

In the circuit of 2PCsha, the oracle and user will jointly perform subtraction and division to deobfuscate data provided by the user. Then both get the intermediate wires and run the SHA256 circuit. On average, the run-time and communication overhead in 2PCsha for different message sizes are shown in Table III. And each part is also divided into the setup phase and the online phase to display separately. Clearly, the interactive solution has better performance than non-interactive solution.

VII. CONCLUSION

This paper presents CertOracle, a management scheme that provides long-term and self-sovereign certificate to solve the problem of traditional certificate expiration and privacy leakage. Specifically, CertOracle allows users to encrypt the private data of the digital certificate by themselves, ensuring that the encrypted certificate is uploaded to blockchain authentically via oracle. Therefore, the blockchain-based identity system can achieve privacy, unforgeability and fine-grained verification. Meanwhile, the uploading method of the certificate can be selected according to the latency of the network between the oracle and the user. The implementation and experiments of CertOracle show that both non-interactive proof and interactive proof protocols can satisfy the requirements of the long-term and self-sovereign certificates.

ACKNOWLEDGMENT

This work was in part supported by Guangdong KeyR&D Plan2020 (No. 2020B0101090002), National Natural Science Foundation of China (Grant No. 61932011), Guangdong Basic and Applied Basic Research Foundation (Grant No. 2019B1515120010), Guangdong Key Laboratory of Data Security and Privacy Preserving (Grant No. 2017B030301004), National KeyR&D Plan2020 (No. 2020YFB1005600), National Joint Engineering Research Center of Network Security Detection and Protection Technology(Grant Nos. 2016B010124009), Guangdong Provincial Special Funds for Applied Technology Research and Development and Transformation of Important Scientific and Technological Achieve (2017B010124002).

REFERENCES

[1] Barker E, Dang Q. Nist special publication 800-57 part 1, revision 4[J]. NIST, Tech. Rep, 2016, 16.

[2] Christopher Allen. The Path to Self-Sovereign Identity [EB/OL]. http://www.coindesk.com/pah-self-sovereign-identity, 2018-08-19/2022-05-17

[3] Manu Sporny, Dave Longley and David Chadwick. Verifiable Credentials Data Model 1.0, 19 Nov 2019. [Online]. Available: https://www.w3.org/TR/verifiable-claims-data-model/. [Accessed: 27 Apr 2020]

[4] Heiss J, Eberhardt J, Tai S. From oracles to trustworthy data on-chaining systems[C]//2019 IEEE International Conference on Blockchain (Blockchain). IEEE, 2019: 496-503.

[5] Signing E U C. TLS-N: Non-repudiation over TLS Enabling Ubiquitous Content Signing[J].

[6] Zhang F, Maram D, Malvai H, et al. DECO: Liberating web data using decentralized oracles for TLS[C]//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. 2020: 1919-1938.

[7] D. Maram et al., "CanDID: Can-Do Decentralized Identity with Legacy Compatibility, Sybil-Resistance, and Accountability," 2021 IEEE Symposium on Security and Privacy (SP), 2021, pp. 1348-1366, doi: 10.1109/SP40001.2021.00038.

[8] Zhang F, Cecchetti E, Croman K, et al. Town crier: An authenticated data feed for smart contracts[C]//Proceedings of the 2016 aCM sIGSAC conference on computer and communications security. 2016: 270-282.

[9] Lerman L, Bontempi G, Markowitch O. Power analysis attack: an approach based on machine learning[J]. International Journal of Applied Cryptography, 2014, 3(2): 97-115.

[10] Oraclize: Understanding oracles. https://blog.oraclize.it/understanding-oracles99055c9c9f7b, accessed 23 Sep 2017.

[11] Breidenbach L, Cachin C, Chan B, et al. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks[J]. 2021.

[12] Peterson J, Krug J. Augur: a decentralized, open-source platform for prediction markets[J]. arXiv preprint arXiv:1501.01042, 2015, 507.

[13] Delignat-Lavaud A, Fournet C, Kohlweiss M, et al. Cinderella: Turning shabby X. 509 certificates into elegant anonymous credentials with the magic of verifiable computation[C]//2016 IEEE Symposium on Security and Privacy (SP). IEEE, 2016: 235-254.

[14] Pittalia P P. A comparative study of hash algorithms in cryptography[J]. International Journal of Computer Science and Mobile Computing, 2019, 8(6): 147-152.

[15] Lindell Y, Riva B. Cut-and-choose Yao-based secure computation in the online/offline and batch settings[C]//Annual Cryptology Conference. Springer, Berlin, Heidelberg, 2014: 476-494.

[16] Schneider T, Zohner M. GMW vs. Yao? Efficient secure two-party computation with low depth circuits[C]//International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2013: 275-292.

[17] Pullonen P. Actively secure two-party computation: Efficient beaver triple generation[J]. Instructor, 2013.

[18] Boura C, Gama N, Georgieva M, et al. Chimera: Combining ring-lwe-based fully homomorphic encryption schemes[J]. Journal of Mathematical Cryptology, 2020, 14(1): 316-338.

[19] Chillotti I, Gama N, Goubin L. Attacking FHE-based applications by software fault injections[J]. Cryptology ePrint Archive, 2016.

[20] Chillotti I, Gama N, Georgieva M, et al. TFHE: fast fully homomorphic encryption over the torus[J]. Journal of Cryptology, 2020, 33(1): 34-91.

[21] Demmler D, Schneider T, Zohner M. ABY-A framework for efficient mixed-protocol secure two-party computation[C]//NDSS. 2015.

[22] Lenstra A K. Key length. Contribution to the handbook of information security[J]. 2004.