Received 12 February 2024; revised 1 March 2024; accepted 19 March 2024. Date of publication 25 March 2024; date of current version 1 May 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3381546

A Blockchain-Based Approach for USIM Management in Mobile Networks

MAEDE HOJJATI[1], ARIAN ARABNOURI[1], ALIREZA SHAFIEINEJAD[1], AND HALIM YANIKOMEROGLU[2] (Fellow, IEEE)

[1]Department of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, Iran
[2]Department of System and Computer Engineering, Carleton University, Ottawa, ON K1S 5B6, Canada

CORRESPONDING AUTHOR: A. SHAFIEINEJAD (e-mail: shafieinejad@modares.ac.ir)

ABSTRACT Universal Subscriber Identity Module (USIM) is an essential part of the mobile network mainly for providing identification and authentication of the subscriber. The activation and deactivation of USIMs are the two most critical services that must be supported by Mobile Network Operators (MNOs). The current solutions suffer from several limitations such as the lack of round-the-clock services and the presence of a single point of failure. In this paper, we propose a blockchain-based scheme for USIM management. Each MNO creates its own smart contract and publishes its address to subscribers. Subscribers can then directly submit their requests by registering a transaction that invokes a specific function of the smart contract. The proposed scheme provides an anytime-anywhere service while at the same time it leverages the benefits of blockchain technology, such as a decentralized architecture that prevents Denial-of-Service (DoS) attacks, as well as a secure auditable log and payment using cryptocurrency. Moreover, we provide a security proof for the scheme through formal verification. Our results demonstrate that our scheme ensures subscriber privacy while providing mutual authentication among participants. Finally, our evaluation on the Ethereum blockchain confirms the efficiency of the scheme in terms of both transaction and execution costs.

INDEX TERMS Mobile networks, blockchain system, formal verification, USIM management.

I. INTRODUCTION

In GSM (Global System for Mobile communication), each subscriber is identified by the use of a Subscriber Identity Module (SIM) [1]. In third generation mobile networks known as UMTS (Universal Mobile Telecommunications System), the terminology was changed and the specifications were split into UICC [2] (Universal Integrated Circuit Card) and USIM [3] (Universal Subscriber Identity Module) specifications. While in GSM, SIM refers to both software and hardware, in UMTS, UICC generally points to a multi-application hardware platform that provides hosting for SIM and/or USIM as applications.

A USIM is a removable smart card for mobile phones which has a critical role in identification and authentication of subscribers. It is a tamper-proof device which provides a managed platform for storing operator specific configurations and subscriber related data. The deployment a USIM card with new services into operational mode, requires a process involving multiple parties such as the card manufacturer, the operator, operator agents or representatives and subscribers. In summary, the USIM lifecycle management process can be described as follows [4]:

1. Planning
2. Ordering
3. Production
4. Test
5. Preactivation
6. Logistics
7. Activation
8. USIM card usage
9. Deactivation

In the first step, MNO designs a USIM based on its criteria and then orders it to a vendor by means of an input file. It contains the profile information, USIM card serial number denoted by ICCID (Integrated Circuit Card Identification

Number) and the IMSI (International Mobile Subscriber Identity). In some cases, such as prepaid cards, the MSISDN (Mobile Station International Subscriber Directory Number) is further identified.

In the third step, production and personalization are done in which the USIM manufacturer creates all secret data, such as PIN codes and OTA (Over-The-Air) keys, and writes them into the card's memory. Further, an *output file* is generated, containing all necessary data in a suitable format which can be fed into the operator's network entities.

Tests are conducted to ensure that each USIM functions properly before shipment to the customer. After successful verification, the pre-activation phase begins in which the output file is securely loaded into the operator's network systems including:

- UDM (Unified Database Management),
- AUSF (Authentication Server Function),
- CCMS (Customer Care and Billing Systems) and
- OTA (Over-The-Air) systems.

The above systems are dedicated for 5G networks. For other networks, some of systems are different. More specifically, the first system is HLR (Home Location Register) and HSS (Home Subscriber Server) in 3G and 4G, respectively. Furthermore, the second system is replaced with AuC (Authentication Centre). in both 3G and 4G.

Then USIM cards are shipped from the card vendor to the operator and distributed to the operator's shopping agents. Before delivering USIM to the subscriber, the activation must be done. In this phase, the MSISDN field is assigned by one of the subscriber's owned phone numbers and then the card is activated in all systems.

Usually, each MNO has a specific customer activation tool which is remotely accessed by authorized agents using an online connection to MNO's customer management system. This tool allows for card activation when the subscriber has obtained a new USIM card. The operator agent is responsible for identification and authentication of the subscriber as well as verifying that the phone number assigned to the MSISDN is owned by the subscriber.

While the USIM card has no validity period to define the end of life, deactivation is an essential operation for lost, stolen, or malfunctioning cards. Deactivation causes the USIM card to be disabled in all systems, but usually, the card data remains in the systems for tracking its history. Deactivation may be initiated by the MNO to replace old USIM cards. In fact, most deactivations are done due to upgrades, in which subscribers with old USIM cards are encouraged by the operator to exchange them for more recent versions of USIM to enable new services.

It should be noted that both activation and deactivation are sensitive services that must be done securely. The lack of sufficient security in the design and deployment of these services can lead to unintended USIM deactivation and activation, which in turn can pose numerous threats against subscribers, such as identity theft, compromised social accounts, privacy violation, and fraud.

Current solutions for USIM activation are dependent on policies established by each country and imposed to the mobile operators. Based on the subscriber identification method, activation is done: 1) by physical document in the store of service provider, 2) by video identification, 3) by physical document in post office and 4) online identification by electronic ID (eID) card. The first way is only good for the provider that facilitate subscribers with physical stores. Although the second solution is preferred to the first method, it will be an offline service if it encounters human intervention for committing the verification. Furthermore, it is usually limited due to age restrictions, legal restrictions, privacy concerns or foreign citizenship. The fourth solution requires eID infrastructure which in most countries are still in the early stages of implementation.

In this paper, we propose a PKI-based solution to integrate blockchain technology into the design of the framework that implements USIM services. This scheme is able to overcome the single point of failure problem as well as providing an automatic platform to support round-the-clock services.

Blockchain, originally known as the underlying technology of Bitcoin [5], has evolved to have applications beyond accounting. Its main characteristics include decentralization, immutability, transparency, auditability, anonymity, and the ability to bring trust to a network. Platforms such as Ethereum [6] and Binance smart chain [7] enable execution of decentralized applications created by third-parties, and thereby bringing dynamicity in a wide range of applications. This capability is known as *smart contract* in blockchain environment.

Due to the significant increase in USIM usage, it is crucial to have an efficient solution for USIM management. In our proposed scheme, each MNO creates its own smart contract and publishes its address to inform subscribers. This allows subscribers to directly communicate with the MNO by registering a transaction that invokes a specific function of the smart contract. It enables an anytime-anywhere service that is independent of the subscriber's location or the time of service. The only requirement is that each subscriber must have an electronic ID for mutual authentication with the MNO.

Our scheme aims to preserve subscriber privacy while at the same time preventing DoS attacks on the MNO's customer management system by keeping it inaccessible to attackers. It provides non-tampered and auditable logs of incoming requests, including USIM activation and deactivation. To the best of our knowledge, this research represents the first work on a 5G customer management system based on blockchain and serves as a foundation for efficient USIM lifecycle management. The main contributions of this paper can be summarized as follows:

- The scheme supports cross-domain authentication of a user to different MNOs and interoperation of them.
- The scheme is based on blockchain which provides high level of security controls including protection against malicious MNO and DoS attacks and thereby solving the single point of failure problem. Further, the smart contract function checks the freshness of incoming request to avoid replay attack.
- The security of the proposed scheme has been proven using formal verification, ensuring the secrecy of credentials information and authentication of subscribers and MNO to each other.
- The efficiency of the proposed smart contract in terms of GAS costs has been assessed.
- The opportunity to pay the charge of service with cryptocurrency instead of national currency.

The rest of this article is organized as follows. In Section II, we review the related work. The required backgrounds are given in Section III. The system model and design goals as well as threat model are presented in Section IV. The proposed scheme is introduced in Section V. The security proof of the scheme is explained in Section VI. The performance of the scheme is discussed in Section VII. Finally, Section VIII concludes the paper.

II. RELATED WORK

To the best of our knowledge, there is currently no published literature discussing the benefits of blockchain for USIM management in mobile networks. In this section, first we discuss the current solution employed by MNOs for subscriber identification for USIM activation/deactivation. Then, we have focused on a set of closely related topic discussing benefits of blockchain in IoT device management. More specifically, we discussed the schemes that uses blockchain for identity management [8] and [9], monitoring the devices in supply chain management [10], [11], [12] and [13], privacy preserving management [14] and [15], creation and handling task [16], bidding purchase management [17], service provisioning [18], interbank customer management [19] and secure firmware updating of IoT devices.

A. USIM ACTIVATION STATE-OF-THE-ART

The type of identification can vary depending on the specific operator and country. Based on the subscriber identification method, one of the followings is utilized:

- Activation in the store of service provider,
- Activation by video identification,
- Activation in post office and
- Online activation by electronic ID (eID) card.

The first three methods need physical documents of subscriber including government-issued ID, international passport, address verification, proof of residency or a combination of them while the last one only requires an eID card.

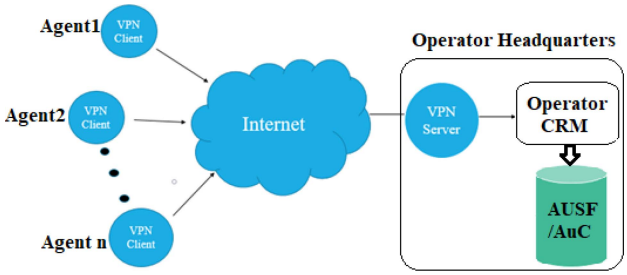

FIGURE 1. Typical connection of operator's agents to CRM.

The first way is the most common way and is good for the provider that facilitate subscribers with physical stores. Each MNO assigns subscriber identification to its agents that accept user request after a rigorous identification. Since the service area of an MNO usually covers an entire state, authorized agents are distributed across different geographical locations. Thus, secure communication of agents with operator customer management system, is a serious challenge. A typical solution used by MNOs, as shown in Fig. 1, is based on a Virtual Private Network (VPN).

For example, "Hamrahe Aval", the first mobile operator in Iran with over 75 million active subscriptions, has approximately 5000 agents throughout the entire provinces of Iran [20]. Each agent establishes a connection to the VPN server of "Hamrahe Aval" to be able to communicate with the Customer Relationship Management (CRM) system. To have a higher security level, each agent is equipped with a USB token which acts as a secure memory to store the credentials required for connecting to the VPN server. At the same time, each agent is restricted to connecting only from a fixed-point location identified by either a specific IP or MAC address.

This architecture exhibits the lack of round-the-clock service (supporting service only during business hours), a significant amount of human resources to manage the agents, human intervention and thereby possibility of human error and/or collusion between subscriber and agent and single point of failure for VPN server in the first layer and the CRM in the second layer.

The second solution is suitable for the operators without physical stores while at the same time facilitate subscribers to purchase a USIM card from any sales point and check the ID through a video connection. Some MNOs may utilize automated systems or AI-powered chatbots to verify user identities during video calls, while others may have human agents or moderators who review and confirm identities. The subscriber should upload his/her ID documents and waits for video call appointments. After a primary verification, the subscriber joins the video call and shows his/her original documents to the video chat agent. Although it is preferred to the previous method, it will be an offline service if it encounters human intervention for committing the verification.

The third solution is suitable for users that online identification is not applicable or not accepted. It is occurred due to age restrictions, legal restrictions, privacy concerns or foreign citizenship.

The fourth solution allows citizens to prove and verify their identity using their eIDs and connecting with their identity provider when requesting on-line USIM activation [21]. Although this provides anytime-anywhere service, it requires eID infrastructure which in some countries such as Estonia and Belgium are well-established and widely used, while others are still in the early stages of implementation. Estonian and Belgian eID are national deployment of eIDAS regulation [22] which is the European directive about electronic identities. Estonian eID [23] is part of "e-residency" program which includes a PKCS11-enabled NFC-based smart card supports online identification, eiDAS-compliant digital signature including timestamping and electronic. Due to substantial invest of time and money, this solution is less versatile than a "conventional" PKI solution.

Our scheme is independent of any infrastructure while eID-based solutions such as [21] need eID infrastructure. Comparing with the scheme with video identification, our scheme has low latency about few minutes without any human intervention. Further, our scheme is based on blockchain and provides high level of security controls including protection against malicious MNO, DoS attacks and thereby solving the single point of failure problem as well as checking the freshness of incoming request to avoid replay attack. Moreover, since both request and response messages are stored on blockchain, our scheme is forensic-ready. It means that in the case of unintended USIM activation/deactivation, both MNO and subscriber are able to decrypt and track the transmitted message to detect any abusing, for example unauthorized use of a person's certificate.

B. BLOCKCHAIN BASED APPROACH FOR DEVICE MANAGEMENT

The scheme in [8] presents a semi-decentralized IoT identity management framework providing identity creation, transfer of ownership and identity portability among networks visited by the devices. It uses a set of smart contracts that provide the functions of the registrar and management contracts. In [9], the authors proposed an IoT device identification and management in 5G smart grid. They used a hybrid blockchain mechanism based on 5G MEC smart grid, where both public blockchain and private blockchain are deployed on the MEC gateway/server. To facilitate the data searching and extracting, they endeavor to build a blockchain explorer indexed by IoT device identifier.

Arumugam [10] proposed a smart logistics solution, logistics planner and condition monitoring of the IoT devices in the Supply Chain Management area using smart contracts. It aims to achieve accountability, traceability and liability for asset handling across the supply chain by various parties involved in a logistics scenario. In [11], the authors presented a blockchain-solution and implementation using Ethereum smart contracts for monetizing IoT data with automated payment involving no intermediary. They discussed key aspects related to architectural design, entity relations, interactions among participants, logic flow, implementation and testing of the overall system functionality. Zuo and Qi [12] proposed an IoT framework for real-time monitoring and control to increase oil field operation and asset efficiency and safety. The authors in [13] proposed a distributed ledger solution to offer a decentralized, privacy-preserving, and verifiable management of Smart Tags during a product's lifecycle. The solution uses the Ethereum blockchain to mediate interactions between the stakeholders during a product's exchange process in which all involved stakeholders and product consumers can verify the product's authenticity without revealing their identity. The proposed solution provides evidence of the product's origin and its journey across the supply chain while preventing tag duplication and manipulation.

He et al. [14] proposed a privacy-preserving IoT devices management scheme which provides efficient time-bound and attribute-based access and supports key automatic revocation. Feng et al. [15] proposed a privacy-preserving tucker train decomposition, to extract meaningful and underlying data generated by different kinds of devices in a wide range of Industrial IoT (IIoT) applications, over blockchain-based encrypted IIoT data without the involvement of users. It enables IIoT data providers to reliably and securely share their data by encrypting them locally before recording them in the blockchain. It consumes massive resources of fogs and clouds to implement an efficient privacy-preserving tucker train decomposition scheme.

Wickstrom et al. [16] proposed a smart contract solution for creating and handling generic IoT tasks. It enables users to manage IoT devices by utilizing the Ethereum infrastructure for authentication, authorization, and communication. This design allows the devices to function autonomously by interpreting tasks received through smart contract transactions, without any direct human to device interaction. Smart contracts in [17] are used for implementing bidding purchase, contract communication, inventory management, and multi-department joint supervision of medical devices.

Alghamdi et al. [18] proposed a secure service provisioning scheme with a fair payment system for lightweight clients based on blockchain. The scheme uses an incentive mechanism based on reputation. It uses a consortium blockchain with the proof of authority consensus mechanism and smart contracts to validate the services provided to the lightweight clients, transfer cryptocurrency to service providers and maintain their reputation.

Hajiabbasi et al. [19] proposed an automated framework for interbank Know-Your-Customer (KYC) in robot-based cyber-physical banking. A deep biometric architecture was used to model the customer's KYC and anonymize the collected visual data to ensure the customer's privacy.

The authors in [24] presented a blockchain-based national Digital ID Framework to improve digital identity government service as a simple single sign-on for each service. The framework consists of smart contracts which are the core of the identity services and incorporate logic.

Zhang et al. [25] proposed a decentralized and multi factor eID registration and verification using a consortium blockchain. The system consists of client, identity registration, identity verification, electronic signature system and blockchain service module. The registration module reads citizen's identity from the ID card, creating a QR code for encrypted eID by signature system. The QR code is stored on blockchain and decentralized identity storage system. The verification system use identity information along with biometric multi-factor identity verification mode to create a unique ID to identify a citizen.

Argento et al. [26] utilized blockchain for service accountability integrating eIDAS-compliant Public Digital Identity. Accountability is usually achieved by involving a trusted third party (TTP). Since blockchain decentralizes trust, it avoids relying on a single TTP.

III. PRELIMINARIES

A. BLOCKCHAIN AND ITS BENEFITS IN 5G NETWORKS

The motivation behind the combination of blockchain and 5G is to utilize the strengths and characteristics of blockchain to address challenges in 5G and open up new opportunities for services and applications. These challenges arise from the variety of new services in 5G and can be identified in terms of decentralization, transparency, interoperability, performance limitations, security, and privacy. Traditional approaches may not be able to overcome these challenges, and researchers recognize the need for innovative solutions [27].

Several schemes proposed in [28], [29], [30], [31], [32], [33] and [34], aim to improve 5G applications using blockchain. In [28], an authentication protocol based on blockchain was proposed for 5G networks. Blockchain is utilized to provide a public channel for communication between the Home Network (HN) and Serving Networks (SNs), enhancing security and protection against malicious SNs. In [29], a sharing economy system for mobile edge computing is proposed, leveraging blockchain for immutability. Guo et al. [30] proposed distributed and trusted authentication system for different edge-based IoT platforms. The proposed system utilizes blockchain to securely store data and user access information. In [31], a hierarchical access control scheme for the cloud is proposed, which uses blockchain to build a key management system. In [32], the authors proposed an access control and authentication scheme for smart grid. Reference [33] utilizes blockchain to provide a secure communication channel by immutability characteristic of transactions. Reference [34] utilizes blockchain for the secure management of Network Function Virtualization (NF), using Practical Byzantine Fault Tolerance (PBFT) as the consensus mechanism.

B. SMART CONTRACT

A smart contract is a computer program that contains a set of rules agreed upon by the parties involved in the contract. Its implementation is guaranteed and produces a definite output. The concept of smart contracts was initially proposed by Szabo [35]. By enabling automatic execution, smart contracts ensure correct execution while eliminating intermediaries and enabling direct peer-to-peer transactions. This reduces costs by eliminating brokerage fees. Smart contracts are decentralized on all network nodes and are executed automatically. The benefits of smart contracts on the blockchain platform include the immutability of the contract and its information, transparency, reviewability, and accessibility

Ethereum is one of the most popular public networks used to create smart contracts on the blockchain platform. It has attracted attention from prestigious projects due to its lower concentration compared to other blockchain networks like Binance Chain [8]. However, projects that do not require a high level of decentralization might prefer running on other blockchains such as Binance Chain due to Ethereum's high cost and low transactions per second (TPS). Ethereum has its own language, Solidity, which allows developers to create and compile desired smart contracts and run them on the Ethereum virtual machine. The concept of GAS in Ethereum provides a reward mechanism for miners and DoS attacks. Moreover, the use of blockchain as a distributed decentralized database ensures data accuracy, accessibility, and resistance to censorship, information deletion, and double spending attacks.

IV. PROBLEM FORMULATION

In this section, we describe the system model and design goals as well as design goals.

A. SYSTEM MODEL

The framework shown in Fig. 2 consists of the following entities:

Subscribers: End users who send activation/deactivation requests. They are classified into subscribers with and without a certificate. Each subscriber with a certificate, which contains his/her social identifier in the subject field, can directly send the desired request to the MNO.

Blockchain: An entity positioned between the MNO and subscribers, acting as a secure communication channel. Additionally, it allows the MNO to create a smart contract that handles both subscriber requests and MNO responses.

MNO: The home network operator that receives subscriber requests through the smart contract function. It processes the request and, in error-free situations, sends a query to the AUSF/AuC system. The query is insertion and deletion for USIM activation and deactivation, respectively. For USIM upgrading, both the deletion of the previous entry and the insertion of a new entry are performed simultaneously in the AUSF/AuC system.

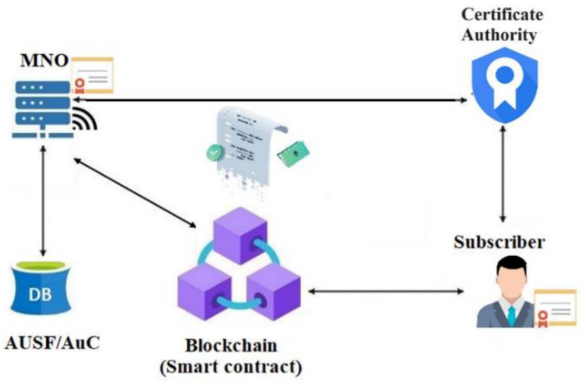

FIGURE 2. Overall architecture of the proposed scheme.

Certificate authority: The entity that is responsible for issuing digital certificate for both MNO and users.

B. THREAT MODEL

First of all, we assume that the interface between subscribers and the MNO is a public channel. It is originated from the fact that we use a public blockchain which anybody can track transactions and scan content of each block.

Secondly, we consider an active attacker model. It means that he/she can inject new messages into the public channel or can save messages for future use to act as either the MNO or a subscriber. It implies that an attacker can eavesdrop on all messages exchanged over it. The attacker may try to act as a malicious MNO to get information from the subscriber.

The attacker may try to conduct impersonation attack. In this case, a legitimate user 'A' who has enough credentials for authentication triggers a successful authentication, pretends to be user 'B', and carries out an USIM activation/deactivation against him/her. The attacker my conduct man-in-the-middle attack for either impersonation attack or acting as malicious MNO. Further, the attacker can request to run multiple instances of the protocol to investigate interleaving attacks.

The only limitation of the attacker is the access to credentials, i.e., he/she has no access to the private key of either MNO or subscriber.

C. DESIGN CONSIDERATIONS

The main ideas of our scheme are as follows:

- We consider blockchain as an interface between the MNO and subscribers who want to send a USIM activation/deactivation request.
- Assuming an insecure channel between the MNO and its subscribers, all messages are encrypted by suitable cryptographic algorithms, i.e., the transaction metadata containing the USIM request/response is always encrypted. Therefore, anyone who scans the blocks of the blockchain is incapable of acquiring any information about the subscriber's sensitive data.

TABLE 1. Notations used in system modeling.

Symbol	Description
SC	Smart Contract
U	Subscriber
MNO	Mobile Network Operator
ID_X	Identifier of participant X
PK_X	Public Key of the principal X
SK_X	Private Key of the principal X
h(.)	Hash function
req_{info}	Request information (encrypted)
$E_{PK_X}(m)$	Asymmetric encryption of message m with public key of participant X
$D_{SK_X}(m)$	Asymmetric decryption of ciphertext c with private key of participant X
$\mathcal{E}_K(m)$	Symmetric encryption of message m with key K
$\mathcal{D}_K(c)$	Symmetric decryption of ciphertext c with key K
$sign_{SK_X}(m)$	Signing message m with private key of participant X
$verify_{PK_X}(m)$	Verify the signature with public key of participant X

- The scheme enforces a rigid access control mechanism for calling the smart contract functions. More specifically, as the owner of the smart contract, the MNO is the only node capable of registering a response transaction.
- The scheme enforces MNO to register the set of valid USIM serial numbers into the blockchain prior to any activation request. It enables the smart contract to verify the validity and freshness of subsequent request and thus aborts invalid and repetitive requests, thereby preventing replay attacks.

V. PROPOSED SCHEME FOR USIM MANAGEMENT

In this section, we introduce our scheme. First, initialization phase is described. Then, the construction of scheme including detail of each phase is explained. Finally, the design of smart contract is presented. Note, that Table 1 summarizes the definitions and symbols used throughout the paper.

A. INITIALIZATION

1) GLOBAL SETUP

In the first step, each user as well as MNO generates a key pair as private and public key of his/her digital certificate (e.g., in X509 format) and signed it by certificate authority. The key pair is used for both encryption and signing of request/response message.

Then, each MNO creates its own smart contract and deploys it on the public blockchain. The MNO publicly publishes the address of this smart contract (e.g., through its official website). Any subscriber who wishes to get a USIM service from the MNO can register a transaction by calling the relevant smart contract function with the required parameters. Using a public blockchain allows the MNO to eliminate the need for infrastructure development, thereby following a pay-per-service business model. As mentioned in the previous section, the smart contract verifies the request for freshness and USIM validation. If these conditions are

not met, the smart contract reverts the transaction and aborts further processing. Otherwise, the request is forwarded to the MNO for final processing. The processing result is also registered in the blockchain by the MNO as a response transaction.

2) REGISTERING A SET OF USIM SERIAL NUMBERS

In this step, the MNO registers the serial numbers of a given set of USIMs in the blockchain. These cards are newly produced USIMs ready to be distributed in the market. This is achieved by calling a specific function of the smart contract. The MNO continuously performs this registration upon receiving demand from the market.

This step is carried out after the preactivation phase and before the shipment of the USIM cards to the MNO's shopping centers. Each subscriber with a valid certificate can purchase a USIM card and activate it for his/her own mobile number.

B. CONSTRUCTION

The overall message sequence is depicted in Fig. 2. It consists of four messages exchanged between subscriber and MNO via smart contract. Each message is dedicated to a specific phase which will be described in more detail in the following subsections.

1) REGISTERING REQUEST BY THE SUBSCRIBER

The details of this phase are shown in Algorithm 1. First, the subscriber generates a 128-bit random key namely K_A, as well as random number R_1 with the same size. Next, subscriber's ID, digital certificate, mobile number along with R_1 are concatenated and encrypted by a symmetric algorithm such as AES using K_A. Let the result be denoted by req_{info}. Then, K_A itself is encrypted twice by an asymmetric algorithm, first with the MNO's public key and the second with subscriber's public key to generate K_1 and K_2, respectively.

Further, a digital signature is generated by subscriber's private key on four-tuple $(req_{id}, req_{info}, K_1, K_2)$ where req_{id} denotes the pair $(USIM_{id}, ServiceType)$. Finally, the subscriber registers a transaction with the five-tuple $(req_{id}, req_{info}, K_1, K_2, req_{sign})$ into the blockchain which invokes a specific function of MNO's smart contract.

To avoid large payload for transaction, we do not directly put the content of the subscriber's certificate into the req_{info} field. Instead, we use a public link to the certificate that is located by the subscriber in a desired public cloud storage. The resulting link usually is a string less than 40 ascii characters.

Further, there is a slight difference in the algorithm when the requested service is a USIM upgrade. In this case, two USIM serial numbers are located in req_{id} field. The first indicates the old USIM which must be deactivated and the second denotes the new USIM that needs to be activated.

To simplify discussion and avoid any loss of generality, we will not address the handling of either upgrade requests or certificate-less subscribers in this paper. However, as mentioned earlier, these can easily be accommodated by making some modifications to the existing scheme.

Algorithm 1 The Subscriber's Request

Input: $USIM_{id}$, $ServiceType$

$K_A \leftarrow$ generating a random 128-bit key
$R_1 \leftarrow$ generating a random 128-bit number
$req_{info} \leftarrow \mathcal{E}_{K_A}(ID_U, Cert_U, \#mob, R_1)$
$req_{id} \leftarrow (USIM_{id}, ServiceType)$
$K_1 \leftarrow E_{PK_{HN}}(K_A)$
$K_2 \leftarrow E_{PK_U}(K_A)$
$req_{sign} \leftarrow sign_{SK_U}(req_{id}, req_{info}, K_1, K_2)$
Subscriber **registers** the transaction $(req_{id}, req_{info}, K_1, K_2, req_{sign})$

2) CHECKING REQUEST AND FORWARDING TO MNO

Upon receiving the request, the smart contract checks the following conditions:

- Freshness of the request.
- Registration of the requested USIM.

The first condition is checked by searching for an entry with req_{id} in previous request records that is currently being processed by the MNO. If such an item is found, the request is rejected and the transaction will be reverted. This effectively prevents reply attacks. For the second condition, the smart contract extracts the USIM identifier from the req_{id} and searches it in registered USIMs. If the USIM identifier is not found, the request is rejected and the transaction will be reverted. Otherwise, the smart contract sets a Boolean flag, called *bRunning*, to true and forwards the request to the MNO for final processing. Setting the flag indicates that the request is currently being processed and any further incoming requests with the same req_{id} will be blocked.

In summary, the smart contract is responsible for preventing replay attacks as well as rejecting requests with invalid USIM.

3) REGISTERING RESPONSE TRANSACTION BY MNO

Upon receiving the request from smart contract, the MNO decrypts K_1 with its own private key to retrieve K_A. Next, the MNO decrypts req_{info} using K_A to obtain ID_U, the subscriber's digital certificate, and the mobile number. Then, the following condition are checked by the MNO:

- Validation of the certificate and its ownership to ID_U by verifying the subject field.
- Verification of request signature by PK_U extracted from the subscriber's certificate.

Violating either of the above conditions prohibits the MNO from further processing and leads to the rejection of the request with an appropriate error code. Otherwise, the process continues by invoking the *Do-Service* function. It first checks the ownership of the mobile number by searching the entry (#mob, ID_U) in AUSF/ AuC. If such an item is

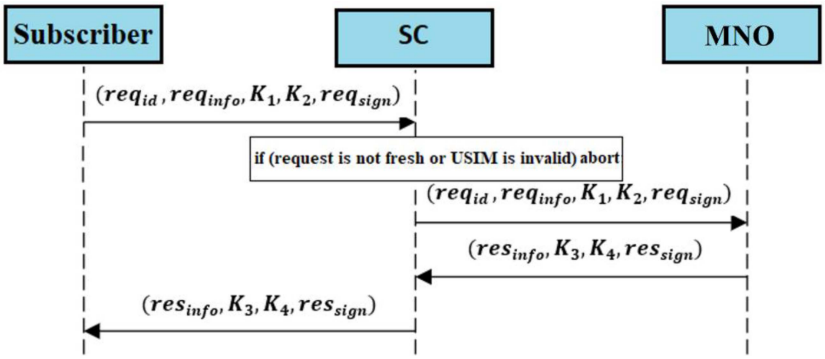

FIGURE 3. The message sequences of the proposed scheme (Smart contract aborts the request if it is not fresh or USIM is invalid).

not found it means that the mobile number does not belon to the subscriber and thus the request is rejected. Now, if the requested service is deactivation, it can be simply done by disabling $USIM_{id}$ in corresponding subsystems. MNO removes the entry ($USIM_{id}$, *#mob*) from AUSF/AuC and also deletes any record containing $USIM_{id}$ in HLR/HSS/UDM, CCMS and OTA repositories. If the requested service is USIM-activation, the MNO further checks the following conditions:

- $USIM_{id}$ was not previously activated for a different mobile number, i.e., there is no record like ($USIM_{id}$, x) in AUSF/AuC for any x.
- The target mobile number is not activated by a different USIM, i.e., there is no record like (x, *#mob*) in AUSF/AuC for any x.

When both conditions are met, the MNO activates $USIM_{id}$ for *#mob* by inserting the entry ($USIM_{id}$, *#mob*) into AUSF/AuC as well as enabling it in HLR/HSS/UDM, CCMS and OTA repositories.

Next, the MNO proceeds to compute the payload of the response transaction by the following actions:

- Selecting a 128-bit random key K_B.
- Computing res_{info} by symmetric encryption of ($result$, R_1) with K_B.
- Asymmetric encryption of K_B by both MNO's and subscriber's public key and putting the results into K_3 and K_4, respectively.
- Generating res_{sign} as digital signature on (res_{info}, K_3, K_4) by MNO's private key.

At the end, the MNO registers the 4-tuple (res_{info}, K_3, K_4, res_{sign}) as the response transaction into the blockchain. It is worth mentioning that regardless of the processing result, the response transaction is always registered by the MNO.

4) REDIRECTING THE RESPONSE TO THE SUBSCRIBER

In this step, the corresponding function checks whether the message sender is the owner of the smart contract. It implies that only the MNO is allowed to register a response transaction to the smart contract. Meanwhile, the flag *bRunning* is set to false, indicating that the request has been completely processed.

5) VERIFYING THE RESPONSE BY THE SUBSCRIBER

Upon receiving the response transaction, the subscriber performs the following actions:

- Verifying res_{sign} by MNO's public key.
- Decrypting K_4 by its private key to retrieve K_B.
- Decrypting res_{info} with K_B to obtain the challenge response and service result.

C. SMART CONTRACT DESIGN

Algorithm 3 presents our smart contract implementation in Solidity, a high-level, object-oriented language for creating smart contracts [36]. There is a pair of data structures, namely *ServReq* and *ServRes,* dedicated to store the request and response information, respectively. Additionally, there are five functions within the contract that perform specific operations of the scheme.

The function *RegisterUSIMSNs* is responsible for registering a range of USIM serial numbers between two 128-bit integers '*beg*' and '*end*'. It inserts each serial number into a map data structure of type Boolean called *RegisteredUSIMs*. This function is called by MNO during the initialization phase.

The next two functions, *ProcessRequest* and *ProcessResponse*, are responsible for handling subscriber's request and MNO's response, respectively, in accordance with phase 2 and 4 of the scheme. Complying with Solidity syntax, the notations have a slight difference compared to those used in algorithms. For example, a request identifier (req_{id}) is denoted by *req_id*.

The former receives the request, including *req_id*, *req_info, K1, K2* and *req_sign* as inputs, and checks two conditions: 1) the registration of the USIM by searching through the map structure *RegisteredUSIMs* and 2) the freshness of request by seeking through *reqMap*, a map data structure that holds information of previous requests. The field *bRunning* of each *ServReq object* shows the state of the request. True means this request is under process, thereby, waiting for MNO response. Against, false indicates the request has been completed.

For an incoming request x identified by *req_id*, if any request with the same identifier and true *bRunning* is found

Algorithm 2 MNO's Response

Input: req_{id}, req_{info}, K_1, K_2, req_{sign}, *ether*

function *DoService*(ServiceType, ID_U, $USIM_{id}$, *#mob*)
if (The owner of *#mob* is not ID_U) **then**
  return ERR-MOB-NUMBER-IS-NOT-BELONG-TO-USER
end if
if (ServiceType = DEACTIVATE) **then**
  Remove the entry ($USIM_{id}$, *#mob*) from AUSF/AuC;
  result ← DEACTIVATION-DONE
else if (ServiceType = ACTIVATE) **then**
  if (an entry like (x, *#mob*) exists in AUSF/AuC) **then**
    result← ERR-MOB-NUM-ACTIVATED-BY-ANOTHER-USIM
  else if (an entry like ($USIM_{id}$, x) exists in AUSF/AuC) **then**
    result← ERR-USIM-IS-CURRENTLY-ACTIVE
  else
    Insert the entry ($USIM_{id}$, *#mob*) into AUSF/AuC;
    result← ACTIVATION-DONE
  end if
end if
return result
end function
($USIM_{id}$, *ServiceType*) ← req_{id}
$K \leftarrow D_{SK_{HN}}(K_1)$
(ID_U, $Cert_U$, *#mob*) ← $\mathcal{D}_K$ (req_{info})
if ($Cert_U$ is not a valid certificate) **then**
  result← ERR-CERT-IS-INVALID
else if req_{sign} is not verified with PK_U **then**
  result← ERR-SIGN-IS-INVALID
else
  result← *DoService*(ServiceType, ID_U, $USIM_{id}$, *#mob*)
end if
$K_B \leftarrow$ gernerating a random 128-bit key
$res_{info} \leftarrow \mathcal{E}_K(result, R_1)$
$K_3 \leftarrow E_{PK_{HN}}(K_B)$
$K_4 \leftarrow E_{PK_U}(K_B)$
$res_{sign} \leftarrow sign_{SK_{HN}}(res_{info}, K_3\ K_4)$
The MNO **registers** the transaction (req_{id}, res_{info}, res_{sign}, K_3, K_4)

Algorithm 3 Solidity Code for Smart Contract Functions

```
pragma solidity ^0.8.7;
contract USIMServingContract
{
  address owner;
  constructor () public {owner = msg.sender;}
    struct ServReq{
    bool bRunning;
    bytes req_info;
    bytes K1, K2;
    bytes req_sign;
    uint eth_val;
}
struct ServRes{
    bytes res_info;
    bytes K3, K4;
    bytes res_sign;
}
mapping (uint => ServReq) private reqMap;
mapping (uint => ServRes) private resMap;
mapping (uint128 => bool) private RegisteredUSIMs;

function RegisterUSIMSNs (uint128 beg, uint128 end) public
{
    require(msg.sender == owner);
    for (uint128 j=beg; j < end; j++) {
      RegisteredUSIMs [j]= true;
    }
}
function ProcessRequest (uint128 req_id, bytes memory req_info, bytes memory K1, bytes memory K2, bytes memory req_sign) public payable
{
    uint128 USIMid = req_id >> 8;
// reject request with unregistered USIM
  if (RegisteredUSIMs[USIMid] && RegisteredUSIMs [USIMid] != true)
    return "Error: Unregistered USIM";
// reject request repeated request
  if (reqMap[req_id] && reqMap[req_id].bRunning) "Error: Repeated request";; reqMap[req_id] = ServReq(true, req_info, K1, K2, req_sign, msg.value);
}
Function ProcessResponse (uint req_id, byte memory res_info, bytes memory K3, bytes memory K4, bytes memory res_sign) public payable{
    require( msg.sender == owner);
    if (reqMap[req_id])
      reqMap[req_id].bRunning = false;
    else
        return "Error: Request is not found";
    resMap[req_id] = ServRes(result, K3, K4, res_sign);
}
function getRequest(uint req_id) public view returns (ServReq memory) {
return reqMap[req_id];
}
function getResponse(uint req_id) public view returns (ServRes memory) {
return resMap[req_id];
}
}
```

in *reqMap*, x is detected as repetitious and accordingly is rejected.

Note that previous requests with false *bRunning* are ignored and the incoming request is forwarded to MNO. This policy is unavoidable, as it allows the subscriber to retry an unsuccessful request after a certain period of time. In summary, this mechanism prevents duplication of active requests for a duration that depends on blockchain network. Since an active request is completed by registration of response transaction, the minimum time for accepting a request with the same *req_id* is longer than the transaction delay.

It means that our scheme on Ethereum blockchain, where the transaction delay is higher than 30 seconds, at least prevents repeated requests within a 30-second interval. This policy effectively mitigates reply attacks. If both conditions are satisfied, a new *ServReq* object is created and initialized

with the input parameters, with *bRunning* set to true. Then, the object is inserted into *reqMap* using the *req_id* as key. Finally, the access fee is received in GAS.

The latter, *ProcessResponse*, is invoked by the MNO to register the response transaction. It searches through *reqMap* for a *ServReq* object with the corresponding *req_id*. If the target object is not found, an error message is returned and the transaction is aborted. Otherwise, *bRunning* is set to false, a new *ServRes* object is created using the input parameters, and added into the *resMap*.

Importantly, the first line of the *ProcessResponse* function ensures that only the owner of the smart contract (the MNO) is allowed to register a response transaction. This is a security mechanism which prohibits an attacker from registering a response transaction, and thus resisting against malicious MNO.

The last two functions, *getReqeust* and *getResponse*, allow the MNO and subscriber, respectively, to retrieve the payload of request and response transactions. The MNO invokes the former at the beginning of phase 3 while the subscriber calls the latter in phase 5. Both of them are defined as *view* functions which implies that they do not modify any data on the blockchain network. As a result, invoking each of them does not impose any payment in GAS on the caller.

VI. SECURITY PROOF

In this section, we will discuss the security of our scheme. First, we will provide a security proof through formal verification. Secondly, we will consider attacks that are not covered by formal verification and discuss our security measures against them.

A. FORMAL VERIFICATION

The message exchanging is modeled by ProVerif [37], a well-known cryptographic protocol verifier. Although, it is basically designed for verification of secrecy and authentication properties, it can also verify additional properties, such as privacy and traceability.

1) THREAT MODEL AND SECURITY REQUIREMENTS

The security goals of our proposed scheme can be divided into two parts: authentication of the participants and secrecy of sensitive data. For the authentication aspect, we aim to provide mutual authentication between subscribers and the MNO. For the secrecy aspect, we investigate the following properties:

- Confidentiality of subscriber's credentials including, temporary and private key, i.e., K_A and SK_U.
- Confidentiality of MNO's credentials including key and secret key, i.e., K_B and SK_{MNO}.
- Confidentiality of subscriber's sensitive data including his/her social identifier, digital certificate and mobile number, i.e., ID_U, $Cert_U$ and #mob.
- Confidentiality of the processing result of the service done either acceptance or rejection of the request as well as its error code, i.e., *result*.

The secrecy of ID_U, $Cert_U$ and #mob aims to preserve the privacy of subscribers. Regarding capability of the attacker, we can refer to the following features:

- The interface between subscribers and the MNO is considered as a public channel. It implies that an attacker can eavesdrop on all messages exchanged over it.
- An active attacker model is assumed. He/she can inject new messages into the public channel or can save messages for future use to act as either the MNO or a subscriber.
- The attacker can request to run multiple instances of the protocol to investigate interleaving attacks.

The only limitation of the attacker is the access to credentials, i.e., he/she has no access to the private key of either MNO or subscriber.

2) PROTOCOL MODELING

The model has three main parts: declarations, process macros, and the main process. For the subscriber and the MNO, we consider the generation of a public and private key pair. This key pair will be used for both message encryption and decryption as well as signature generation and verification.

The declaration part includes four sections: user-defined types, objects, constructors and events. In the first one, *privateKey*, *publicKey* and *SessionKey* indicate the private key, public key and encryption key type, respectively. In the second part, the first statement declares *bcChannel* as a communication channel between subscribers and the MNO. The concept of a free statement is similar to global scope in programming languages; that is, free names are globally known to all processes. The object definition is followed with an access method denoted by either [public] or [private] keywords. Each object which is unknown to the attacker must be declared private, otherwise it is declared public. By default, the access method is public.

The presence of free keyword along with public access method in the definition of *bcChannel* indicates that it is known by the attacker as well as he/she can eavesdrop any message exchanged over the channel. It reflects a realistic assumption about the communication media between MNO and subscribers provided by the public blockchain since a desired node can join the blockchain network and acts as a full node. Each full node has access to transaction data as well as smart contract functions.

The third portion defines function symbols as constructors or destructors. A constructor aims to model a particular primitive of cryptographic protocols. In our scheme, *symEnc*, *asymEnc*, *sign* and *getPK* are the constructors, which respectively model symmetric encryption, public-key encryption, digital signature and returning the public key of a secret key. On the other hand, the relationship between cryptographic primitives is shown by destructors which manipulate terms formed by constructors. In our scheme, *pkiDec*, *symDec* and *checkSign* are the destructors which

Algorithm 4 Type, Object and Function Definitions

```
type SessionKey.
type publicKey.
type privateKey.
free bcChannel: channel. (*Public*)
free K_A: SessionKey [private].
free K_B: SessionKey [private].
free SK_U: privateKey [private].
free SK_MNO: privateKey [private].
free mobNum: bitstring [private].
free ID_U: bitstring [private].
free R_1: bitstring [private].
fun getPK(privateKey): publicKey.
fun asymEnc(publicKey, SessionKey): bitstring.
reduc forall priv: privateKey, k: SessionKey; asymDec(priv, asymEnc(getPK(priv), k)) = k.
fun symEnc(SessionKey, bitstring): bitstring.
reduc forall k: SessionKey, m: bitstring; symDec(k, symEnc(k, m)) = m.
fun Sign(privateKey, bitstring): bitstring.
reduc forall priv: privateKey, m:bitstring; checkSign(getPK(priv), Sign(priv, m)) = m.
fun decode(bitstring): bitstring.
event SubscriberRegisterReq (bitstring).
event MNORcvSubscriberReq(bitstring).
event AcceptSubscriber (bitstring).
event MNORegisterRes(bitstring).
event SubscriberRcvMNORes(bitstring).
```

Algorithm 5 The Subscriber Process

```
let Subscriber(ID_U,USIM_id, mobNum: bitstring, PK_MNO:
publicKey, SK_U:privateKey)=
let K_1 = asymEnc(PK_MNO, K_A) in
let K_2 = asymEnc(getPK(SK_U), K_A) in
let req_info = symEnc(K_A, (ID_U, mobNum, R_1)) in
let Sig_U = Sign(SK_U, (USIM_id, req_info, K_1, K_2)) in
event SubscriberRegisterReq((USIM_id, req_info, K_1, K_2, Sig_U));
out (bcChannel, (USIM_id, req_info, K_1, K_2, Sig_U));
in (bcChannel, (res_info: bitstring, Sig_MNO: bitstring, K_3:
bitstring, K_4: bitstring));
let Key_B = asymDec(SK_U, K_4) in
let response = symDec(Key_B, res_info) in
let Dec_R_1 = decode(response) in
let (=PK_MNO, verify_res:bitstring) = checkSign(PK_MNO, Sig_MNO) in
if (verify_res =(res_info, K_3, K_4) && Dec_R_1=R_1) then
event SubscriberRcvMNORes((USIM_id, res_info, Sig_MNO, K_3, K_4)).
```

respectively model public-key decryption, symmetric key decryption and signature verification algorithms. The last portion defines a set of events described later.

Instead of encoding our scheme in a single main process, we use sub-processes to declare interactions between participants. Each process emulates the action of the corresponding participant to the occurred events. The proposed scheme has three active participants: the subscriber, MNO, and smart contract. However, only two macros are defined, respectively for the subscriber and MNO. In our model, these processes respectively simulate the behavior of the subscriber and MNO defined by Alg. 1 and Alg. 2. We do not model the smart contract as a process macro since it does not have any credentials and does not perform any cryptographic operations. Its main operations are limited to searching through a dynamic map structure and inserting a new item. As mentioned in the declaration part, we model the blockchain as the communication channel *bcChannel*. This simplifies our model but at the same time causes some security features, such as resistance to replay attacks, to not be proven by formal verification.

The subscriber process is shown in Algorithm 5. The subscriber starts the scheme by encrypting the temporary key K_A using both MNO's and his/her public key. Then, he/she puts the result, K_1 and K_2, along with *req_info* on the interface *bcChannel*, and waits for MNO's response. At the same time, he/she triggers a *SubscriberRegisterReq* event to denote reaching a checkpoint in the scheme.

Upon receiving a response from the MNO, the subscriber decrypts K_4 by his/her own private key to recover Key_B and be able to decrypt *res_info*. Then, after successful verification of the signature and challenge response, a *SubscriberRcvMNORes* event is triggered by the subscriber.

On the other hand, the MNO process, shown in Algorithm 6, starts by receiving a request from the subscriber. Next, it triggers a *MNOReciveUserReq* event. After decrypting *req_info* and successful verification of the subscriber's signature, it triggers another event of type *AcceptSubscriber*. It then computes the response message by encrypting *result* into *res_info*. Finally, it triggers a *MNORegisterRes* event and puts *res_info* along with the key materials on the *bcChannel*. Note that MNO process is a simplified version of Algorithm 2, i.e., ignores non-security checking such as ownership of *mobNUM* by ID_U.

The main process is the starting point of the verification. It initializes the scheme by setting up the key materials, communication channel, and then invoking the participant processes. As shown in Algorithm 7, it begins by the keyword *process*, generates private key for both MNO and subscriber along with an identifier for the subscriber. The corresponding public keys are derived by invoking *getPK* constructor and the results are revealed on the public interface *bcChannel*, ensuring that the public keys are accessible to any attacker. Then, it instantiates multiple copies of the subscriber and MNO macros with the corresponding parameters serving as multiple sessions for each principal.

Algorithm 6 The MNO Process

```
let MNO(SK_MNO: privateKey, PK_U: publicKey)=
in (bcChannel, (USIM: bitstring, req_info: bitstring, K1: bitstring, K2: bitstring, Sig_U: bitstring));
event MNORcvSubscriberReq ((USIM, req_info, K1, K2, Sig_U));
let Key_A = asymDec(SK_MNO, K1) in
let (=PK_U, verify_req:bitstring) = checkSign(PK_U, Sig_U) in
if (verify_req=(USIM, req_info, K1, K2))
then
event AcceptSubscriber(K1);
let req_info_dec = symDec(Key_A, req_info) in
let Dec_mobNum = decode(req_info_dec) in
let Dec_R1 = decode(req_info_dec) in
new Key_B: SessionKey;
new result: bitstring;
let res_info = symEnc(Key_B, (result, Dec_R1)) in
let K3 = asymEnc(getPK(SK_MNO), Key_B) in
let K4 = asymEnc(PK_U, Key_B) in
let Sig_MNO = Sign(SK_MNO, (res_info, K3, K4)) in
event MNORegisterRes((res_info, Sig_MNO, K3, K4));
out(bcChannel, (res_info, Sig_MNO, K3, K4)).
```

Algorithm 7 Security Properties and the Main Process

```
query attacker (K_A).
query attacker (K_B).
query attacker (ID_U).
query attacker (mobNum).
query attacker (SK_U).
query attacker (SK_MNO).
query attacker (R1).
query x:bitstring; event(MNORcvSubscriberReq (x)) ==> event(SubscriberRegisterReq (x)).
query x:bitstring; event(AcceptSubscriber (x)) ==> event(MNORcvSubscriberReq (x)).
query x:bitstring; event(MNORegisterRes (x)) ==> event(AcceptSubscriber (x)).
query x:bitstring; event(SubscriberRcvMNORes (x)) ==> event (MNORegisterRes (x)).
process
new USIM: bitstring;
let PK_MNO = getPK(SK_MNO) in out (bcChannel, PK_MNO);
let PK_U = getPK(SK_U) in out (bcChannel, PK_U);
(!Subscriber(ID_U, USIM, mobNum, PK_MNO, SK_U) | !MNO(SK_MNO, PK_U))
```

This provides the benefits of multiple concurrent sessions for the attacker to investigate the possibility of interleaving attacks.

3) SECURITY PROPERTIES

ProVerif attempts to prove that a state which violates a security property is unreachable. A security property is defined by the statement "query attacker(M)" to test the secrecy of term M in the model. Note that M is a ground term, which is likely private and not initially known to the attacker. We consider the following security properties:

- Attack on secrecy of Key_A
- Attack on secrecy of Key_B
- Attack on secrecy of ID_U
- Attack on secrecy of *mobNum*
- Attack on secrecy of SK_U
- Attack on secrecy of SK_{MNO}
- Attack on secrecy of R_1

Further, authentication can be represented by corresponding assertions used to capture relationships between events. This is expressed in the form of "if an event e_1 has been executed, then event e_0 has been previously executed". Annotating the process with events marks important stages reached by the scheme while not affecting other behavior. In our scheme, we define a set of four event pairs:

1. (*MNORcvSubscriberReq*, *SubscriberRegisterReq*)
2. (*AcceptSubscriber*, *MNORcvSubscriberReq*)
3. (*MNORegisterRes*, *AcceptSubscriber*)
4. (*SubscriberRcvMNORes*, *MNORegisterRes*)

The first pair is dedicated to sending a request by the subscriber and receiving it by the MNO. These events are bound together by an inj-event statement in the main process. It means that before the *MNORcvSubscriberReq* event, the *SubscriberRegisterReq* event must be triggered. This binding imposes an order on the sequence of messages exchanged by the subscriber and the MNO.

In a similar way, the rest of pairs are bound together with a query statement. ProVerif can check whether a given binding is satisfied during the execution of the scheme. Thus, reaching the state of the scheme to the point of *SubscriberRcvMNORes* shows that the required messages are exchanged by the participants. Further, the authentication is held if the last message is successfully verified by the subscriber.

4) VERIFICATION RESULTS

The verification results of the secrecy properties of the proposed scheme are summarized as follows:

- RESULT not attacker(Key_A) is true.
- RESULT not attacker(Key_B) is true.
- RESULT not attacker(ID_U) is true.
- RESULT not attacker(*mobNum*) is true.
- RESULT not attacker(SK_U) is true.
- RESULT not attacker(SK_{MNO}) is true.
- RESULT not attacker(R_1) is true.

The results show that the secrecy of the corresponding terms is preserved by the scheme and there is no information leakage for the related sensitive data. Further, for the authentication properties, the verification results are as follows:

- RESULT event(MNORcvSubscriberReq (x)) ==> event(SubscriberRegisterReq (x)) is false.
- RESULT event(AcceptSubscriber (x)) ==> event(MNORcvSubscriberReq (x)) is true.
- RESULT event(MNORegisterRes (x)) ==> event(AcceptSubscriber (x)) is true.
- RESULT event(SubscriberRcvMNORes (x)) ==> event(MNORegisterRes (x)) is true.

The results indicate that all of the authentication properties, except the first one, are satisfied by the scheme. Tracing the verification result for the first message shows that any attacker can act as a subscriber by generating a correct message for the first phase of the scheme. This is originated from the replay attack, since the attacker can eavesdrop on the blockchain interface and collect subscriber requests sent to the MNO. The attacker can pick up one of these requests and send it to the intended MNO. Actually, in our scheme, the smart contract can verify the freshness of request and thereby preventing reply attack. However, since we did not model the smart contract as an active player, the security property related to replay attacks is not satisfied. This causes the verification to generate a false alarm which we ignore it.

B. OTHER SECURITY ISSUES

1) FORGERY AND IMPERSONATION ATTACKS

Each approach to security proof, such as formal verification, has its own shortcomings. The extent of the security proof is mainly dependent on the assumptions about threat and adversarial model. In our formal verification, we assumed an active attacker who has access to the public channel in order to read a message, inject a new message, or change a given message. At the same time, they do not have access to any credentials (e.g., private key of the user) and therefore act as an outside attacker.

However, a significant portion of threats are related to insider attacks, such as forgery and impersonation. In this case, a legitimate user who has enough credentials for authentication triggers a successful authentication, pretends to be a different user, and carries out an attack against him/her. Note that impersonation can also be caused by identity theft or session hijacking, which are the result of bad implementation and are not the subject of our discussion. The defense against impersonation is mainly done by adopting rigorous access control to a given object or data. It is almost infeasible to include access control in the formal model, as well as defining security properties for impersonation attacks.

In our scheme, the conditions checked in the function *DoService* provide resistance to impersonation and forgery attacks. The condition in line 2 implies that #mob is owned by the subscriber who sends the request, thus prohibiting USIM activation/deactivation for a different user. Furthermore, the condition in line 11 states that the current $USIM_{id}$ has not previously been activated for a different #mob, i.e., preventing deactivation of an in-use USIM that is bound to a different subscriber.

2) DOS ATTACKS

Our scheme is resistant against DoS attacks since the MNO only receives incoming requests via the smart contract, not from subscriber. Actually, the need for direct communication between subscriber and MNO is eliminated. Since the smart contract is implemented in a distributed manner, there is no potential for a single point of failure. This mitigates the risk of DoS attacks on the MNO module. However, for a higher level of reliability, a replication mechanism for the MNO module can be taken into account.

Furthermore, the subscriber acts as a simple Registrar Module (RM) that performs a primary check for information provided by the user and finally registers a transaction into the blockchain. There is no need to connect to a centralized database for this checking and registering since the blockchain is used as a distributed ledger to save the request/response data, and the final checking is left to the MNO. The only issue that must be considered in the client program is that it must invoke the valid address of the MNO's smart contract. For this purpose, it is enough to implement the RM as a simple Web application secured by the MNO certificate. To avoid a single point of failure, the RM itself can be replicated regionally, i.e., based on the location of the user, its IP address is connected to a different RM.

VII. EVALUATIONS

A. EXPERIMENTAL PARAMETERS

Registering a request, by invoking *SetRequest*, involves performing a single symmetric encryption, two asymmetric encryptions, and a digital signature generation. We choose AES for symmetric encryption and elliptic curve algorithm for both asymmetric encryption and digital signature. We offer two security modes to accommodate different security levels. In mode 1, we use AES-128 and SECP256R1 for symmetric encryption and asymmetric encryption/digital signature respectively while in mode 2, AES-256 and SECP521R1 are used. It should be noted that SECP256R1 and SECP521R1 are elliptic curves recommended by NIST for 256-bit and 521-bit primes [38].

While mode 2 provides a higher level of security, it also results in larger ciphertext and signature output, leading to higher execution and transaction costs compared to mode 1. GAS, a metric for energy consumption in processing and validating transactions on the Ethereum blockchain, is used to allocate resources on the Ethereum virtual machine. Higher GAS means more work to execute, and the end user registering the transaction pays for it. Table 2 summarizes the experimental parameters which are set in our evaluations.

The request message, in addition to the security level, is also dependent on the length of input parameters. According to ITU standard [39], the identifier of the smart card (ICCID) is encoded as a 10-octet string. Thus, we define USIM_{id} as an array of 10 bytes. Similarly, ITU recommendation E.164 limits MSISDN to a maximum of 15 digits [40], so we consider #mob as an array of 8 bytes. For ID_U as

TABLE 2. Experimental parameters.

Parameter	Length in byte	Description
ID_U	8	16 decimal digits
$USIM_{id}$	10	ICCID is 10-octet string
#mob	8	MSISDN is 15 digits.
Cert	16	A link of 24 base-64 chars
K_A, K_B	16	Security mode 1 (128 bit)
K_A, K_B	32	Security mode 2 (256 bit)
K_1, K_2, K_3, K_4	64	Security mode 1 (2×256 bit)
K_1, K_2, K_3, K_4	132	Security mode 2 (2×521 bit)
req_{sign}, res_{sign}	64	Security mode 1 (2×256 bit)
req_{sign}, res_{sign}	132	Security mode 2 (2×521 bit)

TABLE 3. Running time of basic cryptographic operations.

Operation	Curve	Time on Laptop	Time on Mobile
ECC encryption	P256R1	88 ms	154 ms
ECC decryption	P256R1	43 ms	67 ms
ECC sign	P256R1	0.05 ms	0.13 ms
ECC verification	P256R1	0.01 ms	0.23 ms

national identifier number, we consider 16 decimal digits or equivalently 8 bytes.

To avoid high execution and transaction costs in the *ProcessRequest* function, we suggest storing the certificate file in a public cloud storage like Google Drive and using the URI in req_{info}. This approach allows us to allocate 16 bytes for the certificate link, which is equivalent a string of 24 base-64 characters.

B. PROCESSING TIME ANALYSIS

However, computing request message, needs some calculations among them 2 ECC encryption and 1 ECDSA are significant. Further, for generating response payload at the server, in addition to these operations, a single ECC decryption and verification are required. We measured the time of each operation on a pair of laptop and mobile platform. The former is composed of Windows 10 system (64 bits) with an Intel Core i7-6500U CPU@2.50GHz with 8GB of RAM while the latter is composed of Android 9.0 with Qualcomm SDM730 Kryo 470 CPU@2.2 GHz with the same amount of RAM. The results are shown in Table 3. It is worth noting that the measurement is done by a code written by Python language.

We can see that the processing time for generation of request payload, denoting by $T_{Request}$, is approximately equal to $2T_{ENC}+T_{SIGN}$ in which for a client running on laptop yields $2\times88+0.05 \approx 176$ ms while for a client on mobile platform becomes $2\times154+0.13 \approx 308$ ms. Against, the time for generation of response payload, denoting by $T_{Response}$, is about $T_{DEC}+T_{VERIFY}+2T_{ENC}+T_{SIGN} = 43+0.01+2\times88+0.05 \approx 219$ ms for a server running on laptop platform. For a platform with a lower processing and memory resource than the above assumption, the processing delay can increase. However, we can state the total processing time at both client and server side is less than one second.

C. BLOCKCHAIN TRANSACTION ANALYSIS

Table 4 shows the results of our smart contract implementation on the Ethereum blockchain. The cost of each function depends on both the message size and the number of instructions executed. We can see that the cost paid for security mode 2 is approximately 66% higher than mode 1.

Note that GAS price is not fixed and has a dynamic mechanism to compute the total fee. The Ethereum GAS tracker [41] allows us to monitor GAS price online. The GAS price is periodically updated in 12-second time slots upon generation of a new block. GAS price is expressed in *Gwei* unit which is equivalent to 10^{-9} ETHER (Ethereum cryptocurrency). Tracking the GAS fee over a longer period shows that it fluctuates within a wider range (15 to 37 Gwei). For a lower GAS fee, we can switch to another blockchains such as Binance Smart Chain (BSC) [7]. BSC was initially based on the Ethereum network but now has its own blockchain with the native currency Binance coin (BNB). Following BNB smart chain tracker [42], we can see that the GAS price for BNB Smart Chain is approximately 3 to 5 Gwei, which is lower than Ethereum GAS by a factor of about 5. Moreover, the Binance Coin price versus ETHER is approximately 0.13. This means that the final GAS cost in Ethereum is approximately 38 times more expensive than in BSC. Furthermore, the BSC transaction delay is significantly lower than the Ethereum delay. The difference between Ethereum and BSC primarily stems from the degree of transparency and accuracy. Ethereum provides higher decentralization than BSC but at a significantly higher cost and latency compared to BSC.

D. PUTTING IT ALL TOGETHER

The overall latency of the service from the user's view consists of the time required for encryption and sending the request message, registering the request transaction, decrypting the request by the MNO, creating the response message, and registering the response transaction by the data owner, respectively. Thus, for total delay, we can write:

$$T = 2T_{Trans} + T_{Request} + T_{Response} \tag{1}$$

According to previous section, when running request and response generation on the laptop platform, they will take 176 and 219 milliseconds, respectively. Thus,

$$T = 2T_{Trans} + 0.395\ seconds \tag{2}$$

The delay for registering a transaction is mainly dependent on *Tip*, known as a priority fee, which is an additional fee paid by a user to complete the transaction faster. Ethereum considers three priorities: *low*, *average* and *high* which currently correspond to transaction delay equal to 180, 180 and 30 seconds, respectively [41]. It is worth mentioning that Tip also has a direct impact on GAS prices in Gwei. Faster transactions result in more Gwei being paid for each GAS unit. Therefore, if anybody wants to ensure his/her transaction is processed quickly, they may need to increase the amount of Gwei they are willing to pay for GAS.

TABLE 4. Execution and transaction GAS.

Functions	Mode	Payload size (byte)	GAS		Binance Smart chain			Ethereum		
			Execution Cost	Transaction Cost	Gwei	Total fee in BNB	Trans. delay	Gwei	Total fee in ETHER	Trans. delay
RegisterUSIMSNs	-	32	257,423	279,091	1,609,542	1.61×10^{-3}	5~60 seconds	29,871,440	2.98×10^{-2}	30~180 seconds
ProcessRequest	1	251	302,500	329,304	1,895412	1.90×10^{-3}		35,166,592	3.52×10^{-2}	
	2	455	502,470	532,802	3,105816	3.11×10^{-3}		57,732,576	5.77×10^{-2}	
ProcessResponse	1	235	260,238	286,710	1,640,844	1.64×10^{-3}		30,417,312	3.04×10^{-2}	
	2	409	460,207	490,207	2,851,242	1.85×10^{-3}		52,983,184	5.30×10^{-2}	

This can be adjusted in wallet settings or through transaction settings on some platforms. The amount of Gwei paid for GAS unit will directly affect the speed and priority of a transaction being processed on the blockchain. It is important for users to consider this factor when sending transactions, especially during times of high network traffic when GAS prices may be higher.

Similarly, BSC defines three priorities known as *standard*, *fast* and *rapid* which currently have a delay equal to 30∼60, 10∼30 and 5∼10 seconds, respectively [42]. Thus, the overall delay fluctuates between 60.5 and 360.5 in Ethereum while it varies between 10.5 and 120.5 in BSC based on the priority chosen by the end user at the time of registering the transactionA major challenge in the implementation process is how users or the MNO inform about registering a transaction. In the simplest scenario, the user and the MNO can continuously scan the new blocks in the blockchain and search for the intended request or response message. However, this is not an efficient solution. A more intelligent solution could be deployed by calling a Web service and using a one-time email. The former is suitable for the MNO side while the latter is suitable for deployment on user side. The Web service could be called at the end of the smart contract function *ProcessRequest*, after successful verification of the request. The Web service is not a good solution on the user side since the user usually does not have a fixed address and at the same time it reveals information about the user, which violates user anonymity. Therefore, we propose that each user sends a one-time email address in their request for the smart contract function to be able to redirect the MNO's response to the end user via email service.

VIII. CONCLUSION

In this article, we presented a scheme for automatic USIM management based on blockchain. The scheme allows subscribers to communicate directly with the MNO by registering a transaction on the blockchain. The use of smart contracts enables the scheme to validate the request in terms of freshness and USIM validation, providing transparency, non-repudiation of service, and preventing a single point of failure. Additionally, the security of the scheme has been proven through formal verification, ensuring the secrecy of subscriber credentials and sensitive data, as well as mutual authentication of participants. Moreover, the smart contract was executed on both Ethereum and Binance smart chain, and its execution and transaction costs in GAS fee were measured.

In spite the bandwidth problem of blockchain as well as computational efficiency of smart contracts, our scheme is practical since:

- The payload of transaction is relatively low. Precisely, it is less than 455 and 409 bytes, respectively for request and response messages.
- The smart contract functions do not contain high-load instructions, limited to primitive checking and the insertion of request/response into a map object.
- The execution and transaction GAS are deemed acceptable in most cases.

The proposed scheme demonstrates that a public blockchain platform is capable of replacing certain components of the current CRM system, offering improved service and enhanced security. The use of smart contracts is essential in avoiding a centralized solution that would introduce a single point of failure.

Further, the cost of system maintenance is always a major challenge in legacy solutions, whereas the cost of creating a smart contract is only paid once during the system's lifetime and currently has no time limit. Therefore, schemes like ours, which follow a pay-per-service model, distribute the cost over time and among all subscribers.

Finally, it is possible to present future work that builds upon or extends the current research. That is the proposed scheme has the potential to be integrated with other services such as the 5G authentication and key agreement protocol discussed in [28]. This integration would provide a framework for a wide range of services in 5G and beyond networks.

REFERENCES

[1] "Specification of the subscriber identity module-mobile equipment (SIM-ME) interface; (Release 1)," 3GPP, Sophia Antipolis, France, Rep. 11.11, 2017. [Online]. Available: http://www.3gpp.org/ftp/specs/html-info/1111.htm

[2] "Smart cards; UICC-Terminal interface; Physical and logical characteristics, (Release 15), Version 15.0.0," 3GPP, Sophia Antipolis, France, Rep. ETSI TS 102 221, 2002.

[3] "Characteristics of the Universal Subscriber Identity Module (USIM) application; (Release 15), Version 15.1.0," 3GPP, Sophia Antipolis, France, Rep. TS 31.102, 2018.

[4] J. Cadonau, D. Jayasinghe, and S. Cobourne, "OTA and secure SIM lifecycle management," in *Smart Cards, Tokens, Security and Applications*. Boston, MA, USA: Springer, 2017, pp. 283–304.

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Bus. Rev.*, 2008, pp. 1–9.

[6] V. Buterin, "A next-generation smart contract and decentralized application platform," Etherium, Zug, Switzerland, White Paper, 2017.
[7] "Binance chain." Binance. Accessed: Apr. 10, 2024. [Online]. Available: https://www.binance.com
[8] A. S. Omar and O. Bashir, "Identity management in IoT networks using blockchain and smart contracts," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Soc. Comput. (CPSCom) IEEE Smart Data (SmartData)*, 2018. pp. 994–1000.
[9] D. Wang, H. Wang, and Y. Fu, "Blockchain-based IoT device identification and management in 5G smart grid," *EURASIP J. Wireless Commun. Netw.*, vol. 1, pp. 125–144, May 2021.
[10] S. S. Arumugam, "IOT enabled smart logistics using smart contracts," in *Proc. 8th Int. Conf. Logist., Informat. Service Sci. (LISS)*, Toronto, ON, Canada, 2018, pp. 1–6.
[11] A. Suliman, Z. Husain, M. Abououf, M. Alblooshi, and K. Salah, "Monetization of IoT data using smart contracts," *Inst. Eng. Technol. Netw.*, vol. 8, no. 1, pp. 32–37, 2019.
[12] Y. Zuo and Z. Qi, "A blockchain-based IoT framework for oil field remote monitoring and control.," *IEEE Access*, vol. 10, pp. 2497–2514, 2021.
[13] F. M. Benčić, P. Skočir, and L. P. Žarko, "DL-Tags: DLT and smart tags for decentralized, privacy-preserving, and verifiable supply chain management.," *IEEE Access*, vol. 7, pp. 46198–46209, 2019.
[14] Q. He, Y. Xu, Z. Liu, J. He, Y. Sun, and R. Zhang, "A privacy-preserving Internet of Things device management scheme based on blockchain," *Int. J. Distrib.*, vol. 14, no. 11, 2018, Art. no. 1550147718808750.
[15] J. Feng, L. T. Yang, R. Zhang, and B. S. Gavuna, "Privacy-preserving tucker train decomposition over blockchain-based encrypted industrial IoT data," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 4904–4913, Jul. 2021.
[16] J. Wickstrom, M. Westerlund, and G. O. Pulkkis, "Smart contract based distributed IoT security: A protocol for autonomous device management," in *Proc. IEEE/ACM 21st Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, 2021, pp. 776–781.
[17] X. Yao, X. Zhao, R. He, H. Nie, H. Nie, and S. Sun, "A medical device management system using smart contract on blockchain," in *Proc. 9th Int. Conf. Behav. Soc. Comput. (BESC)*, 2022, pp. 1–5.
[18] T. A. Alghamdi, I. Ali, N. Javaid, and M. Shafiq, "Secure service provisioning scheme for lightweight IoT devices with a fair payment system and an incentive mechanism based on blockchain," *IEEE Access*, vol. 8, pp. 1048–1061, 2019.
[19] M. Hajiabbasi, E. Akhtarkavan, and B. Majidi, "Cyber-physical customer management for Internet of Robotic Things-enabled banking," *IEEE Access*, vol. 11, pp. 34062–34079., 2023.
[20] H. Aval. "Mobile communications company of Iran (MCI)." Accessed: Apr. 10, 2024. [Online]. Available: https://mci.ir/en
[21] (MyProximus, Brussels, Belgium). *Identify Your Prepaid Card*. Accessed: Jan. 1, 2024. [Online]. Available: https://www.proximus.be/en/id_cr_prepidentif/personal/mobile/prepaid-cards/identification-of-your-prepaid-card.html
[22] (Eur. Comm., Brussels, Belgium). *eiDAS Regulation*. Accessed: Jan. 2, 2024. [Online]. Available: https://digital-strategy.ec.europa.eu/en/policies/eidas-regulation
[23] "e-Identity of Estonia." Accessed: Apr. 10, 2024. [Online]. Available: https://e-estonia.com/solutions/e-identity/id-card/
[24] N. Chalaemwongwan and W. Kurutach, "A practical national digital id framework on blockchain (NIDBC)," in *Proc. 15th Int. Conf. Elect. Eng./Electron., Comput., Telecommun. Inf. Technol.*, 2018, pp. 497–500.
[25] Z. Zhang, Y. Sun, P. Huang, and C. Wang, "A blockchain and multi factor fusion based electronic identity registration and verification system," in *Proc. 3rd Int. Conf. Comput. Sci. Manag. Technol. (ICCSMT)*, Shanghai, China, 2022, pp. 431–435.
[26] L. Argento, F. Buccafurri, A. Furfaro, S. Graziano, G. A. G. Lax, and D. Saccà, "Id-service: A blockchain-based platform to support digital-identity-aware service accountability," *Appl. Sci.*, vol. 11, no. 1, p. 165, 2022.
[27] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Blockchain for 5G and beyond networks: A state of the art survey," *J. Netw. Comput. Appl.*, vol. 166, Sep. 2020, Art. no. 102693.
[28] M. Hojjati, A. Shafieinejad, and H. Yanikomeroglu, "A blockchain-based authentication and key agreement (AKA) protocol for 5G networks," *IEEE Access*, vol. 8, pp. 216461–216476, 2020.
[29] M. A. Rahman, M. M. Rashid, M. S. Hossain, E. Hassanain, M. F. Alhamid, and M. Guizani, "Blockchain and IoT-based cognitive edge framework for sharing economy services in a smart city," *IEEE Access*, vol. 7, pp. 18611–18621, 2019.
[30] S. Guo, X. Hu, S. Guo, X. Qiu, and F. Qi, "Blockchain meets edge computing: A distributed and trusted authentication system," *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 1972–1983, Mar. 2020.
[31] M. Ma, G. Shi, and F. Li, "Privacy-oriented blockchain-based distributed key management architecture for hierarchical access control in the IoT scenario," *IEEE Access*, vol. 7, pp. 34045–34059, 2019.
[32] J. Wang, L. Wu, K. K. R. Choo, and D. He, "Blockchain-based anonymous authentication with key management for smart grid edge computing infrastructure," *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 1984–1992, Mar. 2020.
[33] M. Li, L. Zhu, and X. Lin, "Efficient and privacy-preserving carpooling using blockchain-assisted vehicular fog computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4573–4584, Jun. 2019.
[34] G. A. F. Rebello, I. D. Alvarenga, I. J. Sanz, and O. C. M. Duarte, "BSec-NFVO: A blockchain-based security for network function virtualization orchestration," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–6.
[35] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997. [Online]. Available: https://firstmonday.org/ojs/index.php/fm/article/view/548
[36] C. Dannen, *Introducing Ethereum and Solidity*, vol. 1. Berkeley, CA, USA: Apress, 2017.
[37] B. Blanche, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *J. Log. Algebr. Program*, vol. 75, no. 1, pp. 3–51, 2008.
[38] "Elliptic curve cryptography subject public key information," Internet Eng. Task Force, RFC 5480, Mar. 2009, Accessed: Dec. 2022. [Online]. Available: https://www.ietf.org/rfc/rfc5480.txt
[39] "Key establishment between a Universal Integrated Circuit Card (UICC) and a terminal; (Release 7), Version 7.5.0," 3GPP, Sophia Antipolis, France, Rep. TS 33.110, 2008.
[40] *The International Public Telecommunication Numbering Plan*, ITU-Rec. E. 164, Int. Telecommun. Union, Geneva, Switzerland, 2011.
[41] "Ethereum gas tracker." Etherscan. 2024. [Online]. Available: https://etherscan.io/gastracker
[42] "BNB smart chain gas tracker." BscScan. 2024. [Online]. Available: https://bscscan.com/gastracker
[43] B. Lee and J.-H. Lee, "Blockchain-based secure firmware update for embedded devices in an Internet of Things environment," *J. Supercomput.*, vol. 73, pp. 1152–1167, Mar. 2017.
[44] A. Yohan and N.-W. Lo, "An over-the-blockchain firmware update framework for IoT devices," in *Proc. IEEE Conf. Depend. Secure Comput. (DSC)*, 2018, pp. 1–8.
[45] S. Choi and J.-H. Lee, "Blockchain-based distributed firmware update architecture for IoT devices," *IEEE Access*, vol. 8, pp. 37518–37525, 2020.
[46] M. Son and H. Kim, "Blockchain-based secure firmware management system in IoT environment," in *Proc. 21st Int. Conf. Adv. Commun. Technol. (ICACT)*, 2019, pp. 142–146.

MAEDE HOJJATI was born in Tehran, Iran, in 1993. She received the B.S. degree in computer engineering from Arak University in 2015, and M.Sc. degree in computer engineering from Tarbiat Modares University, Tehran, Iran, in 2020. Since 2019, she is a Researcher with the Security Evaluation Laboratory, Tarbiat Modares University. She also works as a penetration testing professional. Her research area includes network security, 5G networks, and blockchain. Her current interests include security and privacy protocols for IoT, blockchain, and 5G security.

ARIAN ARABNOURI was born in Tehran, Iran, in 1990. He received the B.S. degree in computer engineering from Azad Islamic University in 2015, and the M.Sc. degree in computer engineering from Guilan University, Guilan, Iran, in 2020. He is currently pursuing the Ph.D. degree with Tarbiat Modares University. His research area includes formal verification, searchable encryption, 5G networks, and blockchain.

ALIREZA SHAFIEINEJAD received the B.S. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 1999, and the M.Sc. and Ph.D. degrees in electrical engineering from the Isfahan University of Technology, Isfahan, Iran, in 2002 and 2013, respectively. He is an Assistant Professor with the Department of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, where he has been an Assistant Professor since 2015. At Tarbiat Modares University, he leads research in network security and penetration test in Security Evaluation Laboratory. His research interests are in the area of wireless network coding, network security, and design and analysis of cryptography algorithms.

HALIM YANIKOMEROGLU (Fellow, IEEE) received the B.Sc. degree in electrical and electronics engineering from the Middle East Technical University, Ankara, Turkey, in 1990, and the M.A.Sc. degree in electrical engineering and the Ph.D. degree in electrical and computer engineering from the University of Toronto, Canada, in 1992 and 1998, respectively. Since 1998, he has been with the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he is currently a Full Professor. His research interests cover many aspects of wireless communications and networks, with a special emphasis on non-terrestrial networks in the recent years. He has given 110+ invited seminars, keynotes, panel talks, and tutorials in the last five years. He has supervised or hosted over 150 postgraduate researchers in his lab at Carleton. His extensive collaborative research with industry resulted in 39 granted patents. He served as the general chair and the technical program chair of several IEEE conferences. He has also served in the editorial boards of various IEEE periodicals. He received several awards for his research, teaching, and service, including the IEEE ComSoc Fred W. Ellersick Prize in 2021, the IEEE VTS Stuart Meyer Memorial Award in 2020, and the IEEE ComSoc Wireless Communications TC Recognition Award in 2018. He received Best Paper Awards at IEEE Competition on Non-Terrestrial Networks for B5G and 6G in 2022 (grand prize), IEEE ICC 2021, and IEEE WISEE 2021 and 2022. He is currently serving as the Chair of the Steering Committee of IEEE's flagship wireless event, Wireless Communications and Networking Conference. He is also a member of the IEEE ComSoc Governance Council, IEEE ComSoc GIMS, IEEE ComSoc Conference Council, and IEEE PIMRC Steering Committee. He is a Distinguished Speaker for the IEEE Communications Society and the IEEE Vehicular Technology Society, and an Expert Panelist of the Council of Canadian Academies (CCA|CAC). He is a Fellow of the Engineering Institute of Canada and the Canadian Academy of Engineering.