BVPSMS: A Batch Verification Protocol for End-to-End Secure SMS for Mobile Users

Neetesh Saxena *Member, IEEE,* Hong Shen *Member, IEEE,* Nikos Komninos *Member, IEEE,* Kim-Kwang Raymond Choo *Senior Member, IEEE,* and Narendra S. Chaudhari *Senior Member, IEEE*

Abstract—Short Message Service (*SMS*) is a widely used communication medium for mobile applications, such as banking, social networking, and e-commerce. Applications of *SMS* services also include real-time broadcasting messages, such as notification of natural disasters (*e.g.*, bushfires and hurricane) and terrorist attacks, and sharing the current whereabouts to other users, such as notifying urgent business meeting information, transmitting quick information in the battlefield to multiple users, notifying current location to our friends and sharing market information. However, traditional *SMS* is not designed with security in mind (*e.g.*, messages are not securely sent). It is also possible to extract international mobile subscriber identity of the mobile user. In the literature, there is no known protocol that could enable secure transmission of *SMS* from one user to multiple users simultaneously. In this paper, we introduce a batch verification authentication and key agreement protocol, *BVPSMS*, which provides end-to-end message security over an insecure communication channel between different mobile subscribers. Specifically, the proposed protocol securely transmits *SMS* from one mobile user to multiple users simultaneously. The reliability of the protocol is discussed along with an algorithm to detect malicious user request in a batch. We then evaluate the performance of the proposed protocol in terms of communication and computation overheads, protocol execution time, and batch and re-batch verification times. The impacts of the user mobility, and the time, space and cost complexity analysis are also discussed. We then present a formal security proof of the proposed protocol. To the best of our knowledge, this is the first provably-secure batch verification protocol that delivers end-to-end *SMS* security using symmetric keys.

Index Terms—Authentication, Batch Verification, Mobile Subscriber, SMS, Symmetric Key Cryptosystem.

1 Introduction

Cellular and mobile telecommunication industries are one of the fastest growing industries globally, partly due to the capability to provide a wide range of services to the Mobile Subscribers (*MS*s), such as health surveillance [1], health financing and health worker performance [2], Short Message Service (*SMS*)-based web search [3] and end-to-end communications [4], [5]. However, the challenge for the server to handle multiple authentication requests at one time or in a very short time period (*e.g.*, during the first few minutes of a major incident, such as a natural disaster or terrorist attack) is an area that has attracted the attention of researchers in recent years.

- *N. Saxena is with the Department of Computing and Informatics, Bournemouth University, UK. E-mail: nsaxena@ieee.org*
- *H. Shen is with the Department of Computer Science, University of Adelaide, Australia. E-mail: hong.shen@adelaide.edu.au*
- *N. Komninos is with the Department of Computer Science, City, University of London, UK. E-mail: nikos.komninos.1@city.ac.uk*
- *K.-K. R. Choo is with the Department of Information Systems and Cyber Security and Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249-0631, USA. E-mail: raymond.choo@fulbrightmail.org*
- *N. S. Chaudhari is with the Discipline of Computer Science and Engineering, Indian Institute of Technology, Indore, India and also with the Visvesvarya National Institute of Technology, Nagpur, India. E-mail: narendra@vnit.ac.in*

Manuscript received XXXX XX, 20XX; revised XXX XX, 20XX.

1.1 Research Problem

When an *SMS* is sent from one *MS* to another, the information contained in the *SMS* is transmitted as plaintext [6]. *SMS* may also contain confidential information such as PIN number and a link to a login page. Transmission of such confidential information as plaintext over an insecure network can be targeted by an adversary (*e.g.*, intercepting, reading and modifying the *SMS* before it reaches the *SMS*-Center (*SMSC*) [7], [8]. Traditional *SMS* service does not have a mechanism to transmit the message securely from one *MS* to another *MS* or to a group of *MS*s. The *EasySMS* [9] and *SmartSMS* [10] are the only available protocols in the literature that enable secure transmission of *SMS* from one *MS* to another [9]. However, no such protocol exists in the literature that can securely delivers an *SMS* to multiple recipients simultaneously. This is surprising, as in our increasingly interconnected society, there are a number of situations where secure transmission of batch *SMS* can play a crucial role, such as sending urgent business meeting information to two or more employees or to the members of the political parties, military services like simultaneous and quick transmission of secure information in the battlefield, notifying current location to our friends or family members when a person is in trouble, sharing market information, crowd-sourcing information, human flesh search engine of notifying other users about a corrupted public servant by secure *SMS*, and in some cases life-saving (*e.g.*, notifying residents in remote areas of a fast spreading bushfire, an earthquake or a volcano eruption, or notifying all residents and users in the vicinity of an area to stay indoor due to

an ongoing terrorist attack). In many of these applications, we should not compromise on security for the capability for batch dissemination of *SMS*. For example, without an end-to-end (batch) *SMS* security mechanism in place, a malicious attacker could hijack and replace a batch *SMS* from the local authorities with one that will create social unrest (*e.g.*, messages inciting racial hatred). In addition, the protocol should be sufficiently lightweight, suitable for deployment on resource-constrained devices (*e.g.*, limited battery) [11]. We will address such security issues and consider them in the system and threat models.

1.2 Existing Solutions

Several batch verification-based solutions have been designed for different applications. For example, a number of protocols have been proposed for the value added services in Vehicular Ad-hoc Networks (*VANET*) [12], [13], [14], public-private key-based vehicular communication system [15], [16], [17], and digital signatures in batch to achieve high efficiency [18], [19]. Several *SMS*-based wireless protocols [20], [21], [22], [23], lightweight *AKA* [7] and *SMS*-based attacks and their countermeasures are discussed in [8]. Also, the protocols in [24], [25], [26], [27] are designed to provide *SMS* security based on asymmetric key cryptography with the exception in [26]. Other protocols in the literature include [28], [29] designed for the Global System for Mobile Communications (*GSM*), [30], [31], [32], [33], [34], [35] for the Universal Mobile Telecommunications System (*UMTS*), and [36], [37] for the Long-Term Evolution (*LTE*) networks. However, all these protocols do not consider simultaneous multiple authentication requests using *SMS*. Group Authentication and Key Agreement (*AKA*) protocols are also available in the *LTE* network [38], [39]. However, these protocols do not consider *SMS* as a communication medium and require additional cost and storage for a group setup. Recently, a solution for user privacy in mobile telephony was proposed using the predefined multiple International Mobile Subscriber Identities (*IMSI*) for each Universal Subscriber Identity Module (*USIM*) [40]. However, this solution requires a large storage space, generates a huge overhead for pseudo-identities, and utilizes significant bandwidth for sending *IMSI*s to each *MS*.

The *SSMS* protocol [41] and the Secure Extensible and Efficient *SMS* (*SEESMS*) protocol [42], [43] do not provide end-to-end *SMS* security for mobile users, rather they provide security between a mobile user and the server, which is just a subpart of the problem of this paper and also was the same in *EasySMS*. These protocols in addition to the protocol in [44] do not focus the communication and computation overheads, prevention from attacks, and the bandwidth utilization. The additional drawback with *SEESMS* is that it was not suitable for resource constraint devices like mobile phones. There are other existing protocols as well for securing *SMS* between a mobile user and the server: *SecureSMS* [45], *SK-SIM* [46], and *SMSSec* [47], however, they do not provide *SMS* security between the end users. Similarly, the *VAS-AKA* [48] protocol does not provide end-to-end *SMS* security for mobile users, rather it secures the communication between a mobile user and the service-provision server in order to deliver value added services.

There are mainly two approaches of designing authentication system, namely peer-to-peer approach and server-based approach. The peer-to-peer security approach recommended in [49] is based on the public key crytosystem, which requires a public-private key pair for each user. It does not require a central authority or a central server for authentication, but the overall processing is relatively slow in comparison to a symmetric key cryptosystem. On the other hand, [50] and [51] mentioned that the Authentication Server (*AS*) is a better strategy, as it provides faster processing and supports lightweight authentication for a large number of users. However, a single point of failure can be an issue if there is only one *AS* deployed in the system. In this paper, we selected the server-based approach for authenticating user requests as it supports the existing cellular infrastructure, where an *AS* is a part of the Authentication Center (*AuC*).

A literature review suggests that there is no known batch verification-based protocol that provides end-to-end *SMS* security to many *MS*, although we observe that commercially available applications, such as *SMSzipper*, *TextSecure*, *moGile Secure SMS*, and *CryptoSMS* provide the facility to send secure *SMS*. However, there are a number of limitations in these software solutions, such as (i) the need to install them on the phone's memory/memory card, (ii) the need to provide a secret key to the *SMS* recipient, and (iii) the inability to support sending of an *SMS* to many users simultaneously. Moreover, the security of the communications may also be affected by malware installed or vulnerabilities on the client devices. Therefore, a preferred solution is to develop a protocol that provides end-to-end security.

1.3 Our Contribution

In this paper, we propose a secure and efficient batch verification-based *AKA* protocol, hereafter referred to as Batch Verification Authentication and Key Agreement Protocol (*BVPSMS*), which enables the transmission of an *SMS* to multiple recipients at any one time. The *BVPSMS* uses symmetric keys, since symmetric key encryptions are significantly faster than asymmetric key encryptions and as shown in [52], [53], they consume less energy. The proposed protocol has the following contributions:

1) The *BVPSMS* protocol:
   a) provides mutual authentication between the sender *MS* and the Authentication Server (*AS*), and between each recipient MS_i and the *AS*.
   b) maintains message confidentiality and integrity using *AES* with Counter (*AES-CTR*) and Message Authentication Code (*MAC*), respectively, during messages transmission over an insecure network.
   c) allows the sending of only one of n-pieces of the secret code of the key by sender *MS* to each recipient *MS*. It has the following advantages: (i) sending a partial code to each recipient *MS* improves the overall security of the system, and (ii) reduces the total communication overhead generated by the protocol.

2) Our protocol is secure against replay attack, Man-in-the-Middle (*MITM*) attack, impersonation attack, *SMS* disclosure, and *SMS* spoofing.
3) Each user's original identity is kept secret during the authentication over the network. It protects the user against *IMSI* tracing and *ID*-theft attacks.

We compare our protocol with four other related protocols (*ABAKA*, *RAISE*, *SPECS*, and *b-SPECS+*). In a batch authentication when number of requests are 5, 10, 20, 50, 100, and the findings are as follows:

1) During first time (fresh) authentication, *i.e.*, *BVPSMS**, reduces 6.1%, 23%, 12.5%, and 46.52% of the communication overhead as compared to *ABAKA*, *RAISE*, *SPECS*, and *b-SPECS+*, respectively, and is equal of the *BLS* protocol. However, *BLS* does not provide mutual authentication, user privacy, integrity protection, and offers only partial resilience to impersonation attack.
2) During each subsequent authentication, *i.e.*, *BVPSMS***, lowers the communication bandwidth by 79.27%, 89.83%, 80.69%, and 88.2% in comparison to *ABAKA*, *RAISE*, *SPECS*, and *b-SPECS+*, respectively.

In addition, findings from the simulations (*i.e.* execution time, verification time, and re-batch verification time) demonstrate the utility of our protocol in a real-world cellular network deployment.

1.4 Organization

The remainder of the paper is organized as follows. Section 2 describes the system and threat models for *SMS* security. Section 3 presents our proposed protocol. Section 4 presents the reliability analysis of the proposed protocol, a malicious request detection algorithm, and the impact on user mobility. The security analysis and the performance evaluation of the *BVPSMS* protocol are presented in Sections 5 and 6, respectively. Finally, section 7 concludes this work.

2 System and Threat Models

In this section, we present the system and threat models.

2.1 System Model

We introduce a scenario where the *MS* sends an *SMS* to multiple *MS*s simultaneously. Upon receiving the *SMS*, each *MS* sends its authentication request to the *AS* for identity verification of the sender *MS*. The system model allows many such concurrent executions (*e.g.*, several *MS* sending *SMS* to multiple recipients *MS*. A scenario is shown in Figure 1 where multiple *MS*s send their authentication requests to the *AS* for the identity verification of sender *MS* at the same time. The *AS* handles the received authentication requests and authenticates all the *MS*s. The authentication request may be single or multiple. However, it would be uncommon to have only a single request at any point of time. When an *SMS* is sent from the sender *MS* to the recipient *MS* over the *2G/3G* (*GSM/UMTS*) networks, it follows the path shown in Figure 2(a) [54], [55]: Sender *MS*→Base Transceiver Station (*BTS*)→Base Station Controller (*BSC*)→Mobile Switching Center (*MSC*)→*SMS*-Gateway *MSC* (*SMS-GMSC*)→*SMS*-Center (*SMSC*)→*SMS-GMSC*→*MSC*→*BSC*→*BTS*→Recipient *MS*. Similarly, Figure 2(b) and Figure 2(c) show a path of *SMS* transmission over the *SG*s and *IP/IMS* in *4G* (*LTE*) networks. It is challenging for the *AS* to verify and authenticate a large number of *MS*s, based on its capacity to handle requests in an efficient way.

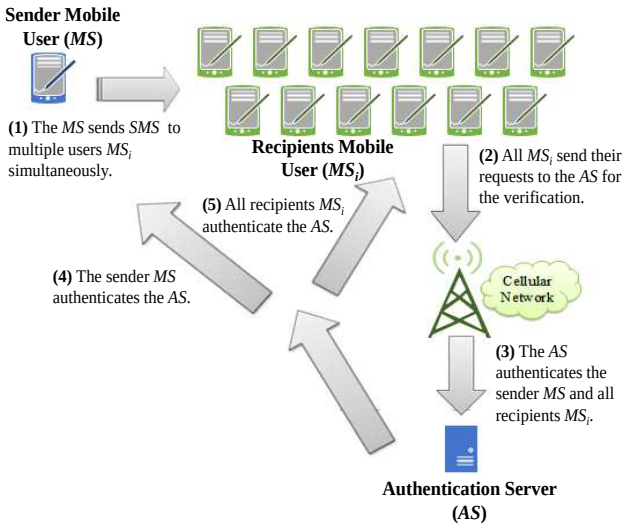
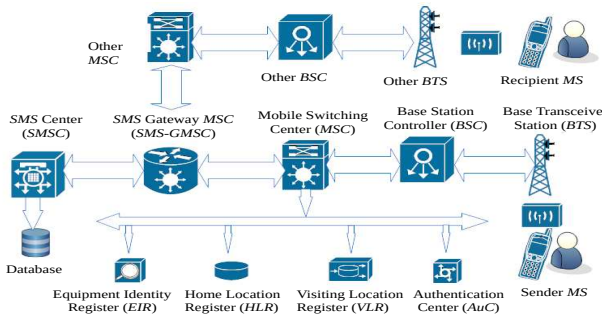


Fig. 1: Batch authentication requests from the *MS* to the *AS*.

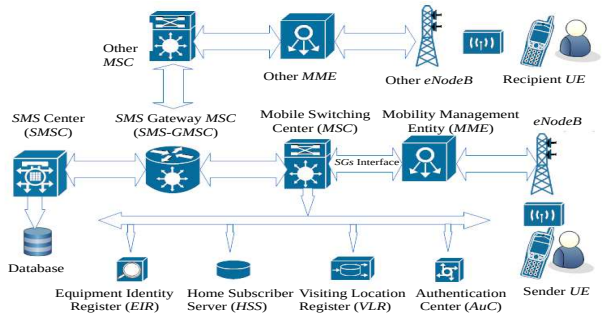
If the server can only handle one request at a time, then it requires a queue to manage all incoming requests. However, managing such a queue will result in increased overheads, time, and cost of authentication. In fact, the approach used for the authentication must be very efficient to handle all the requests in a very short time. To more efficiently handle multiple authentication requests, one solution is to perform a batch authentication for all incoming requests. However, there may be one or more malicious requests generated by the adversary. In such a case, we need to first identify the malicious requests and remove the identified malicious requests from the batch, then perform re-batch authentication. This comes at an additional cost to the re-batch authentication. However, the cost of authenticating each user is reduced. The notations used in the paper are presented in Table 1.

2.2 Threat Model

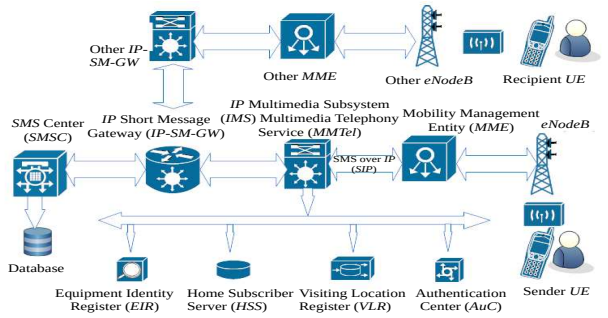
We consider a threat model with three categories of mobile users, namely honest majority, semi-honest majority, and dishonest majority. In the *honest majority* scenario, the legitimate and honest *MS* and the *AS* behaves as per protocol specifications, while a few (no more than half the total) *MS* send incorrect outputs to the *AS* in a *semi-honest MS* scenario. However, in the *dishonest MS* (malicious *MS* to the network) scenario, majority of the *MS* (more than half) send fabricated information to the *AS*. Furthermore, malicious



(a) *SMS* transmission in *2G/3G* (*GSM/UMTS*) system.



(b) *SMS* transmission over *SGs* in *4G* (*LTE*) system.



(c) *SMS* transmission over *IP/IMS* in *4G* (*LTE*) system.

Fig. 2: *SMS* transmission from the sender *MS*/*UE* to receiver *MS*/*UE* in cellular systems.

MS computes the required functions in a probabilistic polynomial time with auxiliary information. We do not consider these scenarios for the *AS*, as malicious *AS* does not have the correct keys in its database. Therefore, we consider only the trusted *AS* scenario where the *AS* always sends correct information to all the *MS*s. We also remark that an adversary can delay some or all the messages between the *MS* and the *AS* under a public channel.

In this paper, we consider two variations of the adversary models: non-adaptive and adaptive variations. In a non-adaptive or static variation of the model, a set of corrupted users are fixed, while in the adaptive variant, the adversary can choose any corrupted users in any numbers during run time. Furthermore, the adversary can choose any input for corrupted users. We also consider *passive as well as active adversaries* in the network.

i) Security and Privacy Attacks, and Integrity Violations:

TABLE 1: Notations

Symbol	Description	Size (bits)
MS	Mobile station referring user	–
UE	User Equipment referring user	–
AS	Authentication server referring *AuC*	–
IMSI	International mobile subscriber identity	128
TID	Temporary identity	128
ReqNo	Request number	8
SK	Shared secret key between *MS* and *AS*	128
DK	Delegation key generated from *SK*	128
H/MAC	Hash/message authentication code	64
$T/T_i/T_1$	Timestamp	64
K	Random number	128
$Y/P/Q/R$	Variable	128
Z	Signature generated by the *MS*	128
SIMcode	SIM card activation code of *SK* key	64
S-Actcode	Sender generated code of the *SK* key	64
Actcode	Recipient generated code of the *SK* key	64
ExpT	Expiry time	64
$f_1()$	*HMACSHA-256* is used to generate *DK*	–
$f_2()$	*AES-CTR* is used to generate *TID*	–
$f_3()$	*HMAC-SHA1* is used to generate *MAC*	–
$E\{\}_{DK}$	Encryption function with *DK* key	–
$D\{\}_{DK}$	Decryption function with *DK* key	–
$\oplus$	Bitwise *XOR* operation	–
$\rightarrow$	Right arrow	–

The threat model describes different scenarios to capture various attacks in which a malicious *MS* can access the authentic information or misguide legitimate *MS*. Since the *SMS* is sent in plaintext, network operators can eavesdrop on the *SMS* content at the *SMSC*. This leads to *SMS* disclosure and spoofing attacks. Currently, Over-the-Air (*OTA*) interface between the *MS* and the *BTS* is protected by a weak encryption algorithm, such as *A5/1* or *A5/2*. Hence, the adversary can compromise the messages in order to capture the information contained in the *SMS*. The unencrypted messages are sent over the Signaling System (*SS7*) networks, which does not secure the transmission medium.

ii) Security Goals: Our security goals are as follows:

1) *Mutual Authentication:* The proposed protocol must provide mutual authentication between each *MS* and the *AS*.
2) *Data Confidentiality and Message Integrity:* These are two key properties to prevent the leakage or abuse of user data.
3) *Other Security Properties:* The protocol should be secured against the following attacks:
   a) *Eavesdropping and Impersonation Attacks:* The adversary can eavesdrop the communication between the user and the server. The adversary may also pretend itself as legitimate user or the server and perform impersonation attacks. The half-open connection requests lead to flood-based Denial of Service (*DoS*) attacks [56]. There are cellular network protocols that protect the system against a *DoS* attack, such as *Secure-AKA* in *UMTS* network [57].
   b) *MITM Attacks:* An adversary can perform *MITM* attack when the *MS* is connected to the *BTS* and eavesdrops the session initiated

by a legitimate *MS*. If *IMSI* is sent in cleartext, the adversary can compromise the system/user by tracing the user. Commercially available software, such as *IMSI* catcher can be used to capture the user's *IMSI* over a weak or unencrypted network.

c) *Replay Attacks:* The attacker may fraudulently delay the conversation between both *MS*, and captures or reuses the authenticated information contained in the previous messages to facilitate or conduct a replay attack.

4) *Session Key Security, Forward Secrecy, and Non-linkability:* It is common practice not to send the session key over the network in a plaintext. The system must also defeat *known key* attacks and maintain forward secrecy. The protocol should be able to handle key generation, transmission, and its usage. The adversary must not be able to link current session information (messages and keys) with previous sessions, *i.e.*, *non-linkability*.
5) *Privacy Preservation and Untraceability:* The original identity of each *MS* must be protected during its transmission over the network. Such *privacy preservation* helps to secure the system against *MITM* attacks and user *untreaceability*.

3 Proposed Protocol: BVPSMS

In this section, we present the proposed efficient and secure batch verification-based protocol *BVPSMS* for end-to-end *SMS* security over an insecure network. The *BVPSMS* protocol is illustrated in Figure 3. The following subsections describe our protocol in detail.

3.1 System Assumptions

We make the following assumptions, similar to the traditional cellular network, for our system implementation:
Assumption 1. An *AS* is deployed at the Authentication Center (*AuC*) similar to the traditional cellular network.
Assumption 2. A Secret Key (*SK*) is stored in the *AS*'s database at the *AuC* as well as on the Subscriber Identity Module (*SIM*) card of the *MS* during manufacturing.
Assumption 3. The *AS* never discloses the stored secret keys to any other entity in the network. Also, it does not illegally reuse the secret key of any one mobile user to other users.
Assumption 4. The process of generating *Actcode* and retrieval of *SIMcode* (discussed later in user registration subsection) is strictly kept secret and not publicly available. This is a realistic assumption as cellular network algorithms and functions are generally considered intellectual property.

3.2 Definition of the Functions Used

Our protocol uses different functions with standard notations, such as $f_1()$, $f_2()$, $f_3()$, and $E/D\{\}$, similar to used by existing cellular network authentication protocols. In the protocol, $f_1()$ and $f_3()$ functions are two different *HMAC* functions to avoid any collision generated with the same input. We also consider *AES* with Counter mode (*AES-CTR*) to implement $f_2()$ and $E/D\{\}$. However, inputs for $f_2()$ and $E/D\{\}$ are different. Modern mobile devices are fairly capable of computing these functions [9]. The structure of these functions as follows:

$f_1()$ ***Function:*** A one-way function, such as one-way hash function *HMACSHA-256*, which takes input message of 512 bits with *SK* key and generates 256 bits of hash code, out of which first 128 bits are used as the *DK* key.

$f_2()$ ***Function:*** Any reversible symmetric encryption function, such as *AES-CTR* where the plaintext and shared key generate the ciphertext, and then ciphertext and the same key are able to produce the original plaintext. The key used in the function is *DK*, derived from the *SK* key at *MS* as well as at *AS*.

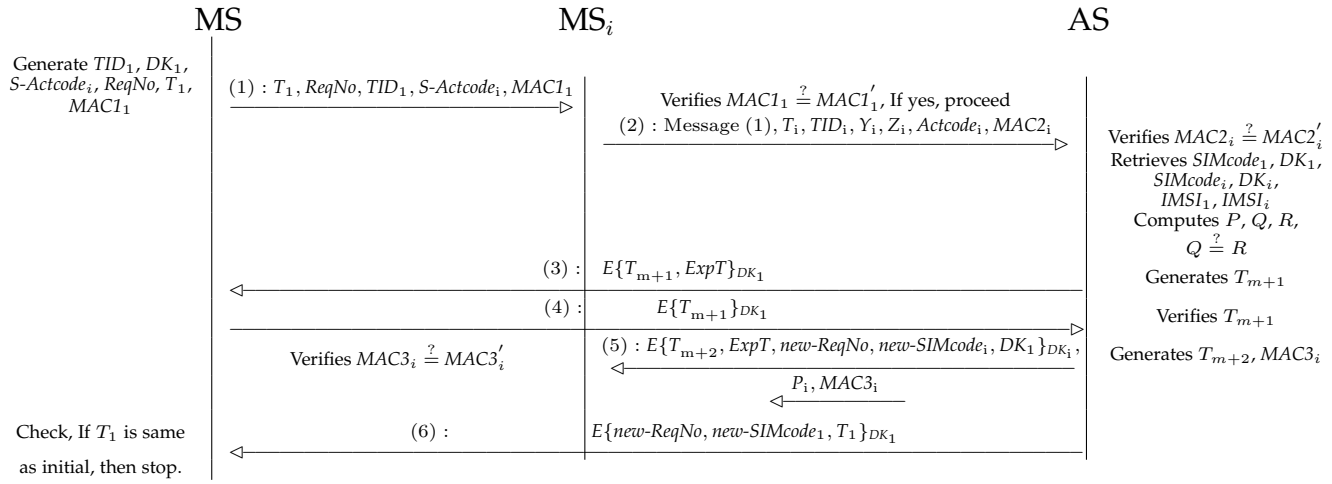
$f_3()$ ***Function:*** It is used to generate *MAC* codes, which can be implemented by a one-way *MAC* function, such as *HMAC-SHA1* that takes as input a multiple of 512 bits message with *DK* key and generates 160 bits of hash code, where the first 64 bits are used as *MAC*.

$E/D\{\}_{DK}$ ***Function:*** It is used to encrypt and decrypt the transmitted messages over the network. *AES-CTR* with *DK* key is used for this purpose. The Modified *AES* (*MAES*) [9] with 256 bits of *DK* key can also be used as an alternative. However, a key expand function is required to generate 256 bits of *DK* key from 128 bits.

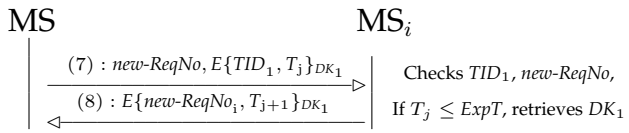
3.3 Detailed Description

Although the protocol is capable of supporting concurrent threads of different *MS*s sending their authentication requests with *SMS*s to several different *MS*s. For simplicity, in this section, we present a scenario where a *MS* sends multiple *SMS*s to different *MS*s. This scenario can be easily extended with multiple sender *MS*s. The physical security (any personal access by the end user/mobile operator/adversary) of the *AS* is assumed secure, similar to the existing traditional cellular networks. Hence, it is almost impossible to extract the secret key *SK* of a mobile user. Readers should not confuse the *AuC* with the *SMSC*. At the *SMSC*, mobile operator can easily access the content of each message. The *AuC* is secured against any personal access, and the keys stored at the *AuC* can only be accessed by the protocol during its execution. Therefore, the *AS* is secure against any personal access.

The proposed protocol uses a temporary identity (*TID*) for each user involved in the authentication process. The idea is to protect the user's original identity (international mobile subscriber identity *IMSI*, and not a mobile number) over an insecure network. A mobile user can be traced by its *IMSI* number. A mobile number is used just to connect with a user, but it is never sent or exchanged over the network. In other words, this mapping to recognize the actual user takes place only with the *IMSI* number. Relevant authorities, such as safety and security department, can trace the original user with the help of the cellular providers, but an attacker or other legitimate users cannot trace a user over the network during communication.



(a) Phase-1: protocol execution for mutual authentication.



(b) Phase-2: subsequent authentications.

Fig. 3: BVPSMS protocol (a) phase-1 (b) phase-2.

We describe our protocol in four different parts: user registration, pseudo-identity generation, protocol initialization, and protocol execution. The protocol maintains message integrity between each *MS* and the *AS* using *MAC*s.

1) User Registration: When a user requests for a new *SIM*, the operator activates *SIM* card by establishing a connection between the *SIM* card and the *AS*. The *AS* generates a random $SIMcode \in \mathbb{Z}_p^*$ (where p is a large prime), stores *SIMcode* in its database as a label to the secret key *SK*, and also sends *SIMcode* to the *SIM* card during first use (*e.g.*, when the card is activated). On receiving *SIMcode*, the *SIM* card stores it in the memory. The *Actcode* is a one-time activation code sent to the *AS* instead of the actual *SIMcode*, when requesting for the authentication. The purpose of this code is to help the *AS* to verify *SIMcode* and retrieve *SK* key from its database that belongs to a user requesting for the authentication. The *AS* sends a random *new-SIMcode* to all involved *MS*s for subsequent authentication request.

In the proposed protocol, when a mobile user activates this module to send an *SMS* to multiple users, an automatic signal is sent to the respective *AS*, which sends a random k to the user's device encrypted by its *DK* key. The user decrypts k and chooses n by its own. The selection of n is based on the average number of *SMS*s dropped by the network per unit time. Although there is no guarantee that an *SMS* will actually be delivered to the recipient, but delay or complete loss of a message is uncommon, typically affecting less than 5 percent of messages [58]. Hence, our scheme uses $n \leq 1.05 \times k$. The generation and transmission of activation code *S-Actcode* by the sender *MS* and retrieval of actual *SIMcode* by the *AS* are motivated by Shamir Secret Sharing Scheme [59] as follows:

The goal is to divide the hash of secret *SIMcode* of the sender *MS* into n-pieces as {*S-Actcode*$_1$, *S-Actcode*$_2$, ..., *S-Actcode*$_i$} (i = 1, 2, ...,n) such that: (i) knowledge of at least k pieces of *S-Actcode*$_i$ helps *AS* in the computation required to generate the final *SIMcode*, say *SIMcode*$_1$'s hash, and (ii) knowledge of any k-1 pieces of *S-Actcode*$_i$ cannot help in the reconstruction of the final *SIMcode*$_1$'s hash (considering all possible values are equally likely). Therefore, the sender *MS* sends *S-Actcode*$_i$ to n-recipients *MS*$_i$. All n-*MS*$_i$ (in the ideal case) or at least k out of n-recipients *MS*$_i$ (in case of error or network failure) forward their *Actcode*$_i$ to the *AS* along with the received *S-Actcode*$_i$ (part of sender *MS*). The *AS* obtains the actual hashed *SIMcode*$_1$ after receiving at least k-*S-Actcode*$_i$. The *AS* will then match the computed hashed *SIMcode*$_1$ with the stored pre-computed hash of *SIMcode*$_1$ of the sender *MS*. Once the hashed *SIMcode*$_1$ is known to the *AS*, it retrieves SK_1 key and derives a delegation key DK_1 of the sender *MS*. This entire process takes k points to define a polynomial of degree k-1 in a finite field $\mathbb{F}$ of size p where $0 < k \leq n < p$, *SIMcode*$_1 < p$, and p is a large prime.

The sender *MS* chooses at random k-1 positive integers $\{b_1, b_2, ..., b_{k-1}\}$ with $b_i < p$, and computes a polynomial $f(x) = b_0 + b_1 x + b_2 x^2 + ... + b_{k-1} x^{k-1}$, where b_0 = *SIMcode*$_1$. The sender *MS* generates n *S-Actcode*$_i$ points (x_i, y_i) as $(i, f(i) \ mod \ q)$ using the Lagrange basis polynomial, where $q > n$, $q > b_i$. On receiving the message (from at least k-recipients *MS*$_i$), the *AS* reconstructs a polynomial by computing $f(x)$ as:

$$f(x) = \sum_{i=1}^{k} y_i l_i(x), \text{ where } l_i(x) = \prod_{\substack{1 \leq j \leq k \\ j \neq i}} (x - x_j)/(x_i - x_j).$$

Finally, the *AS* retrieves the actual *SIMcode*$_1$ (= b_0) from the computed $f(x)$. In our protocol, each recipient *MS*$_i$ also generates its own *Actcode*$_i$ as follows:

At the MS_i: Each MS_i generates $Actcode_i = H(SIMcode_i)$ and is sent to the *AS*. We use first 64 bits of $H()$ function as $Actcode_i$, which is *SHA256*.
At the AS: The *AS* pre-computes $H(SIMcode_i)$ from the stored $SIMcode_i$ for each MS_i, and then verifies $Actcode_i \stackrel{?}{=} H(SIMcode_i)$. Thereafter, the *AS* extracts SK_i key and derives DK_i key by referring $SIMcode_i$ of each MS_i. The delegation keys are special keys that represent the corresponding users and are derived from the original keys. This idea prevents the original keys from been accessed by adversaries over the network. In short, the original keys cannot be derived and comprised by the unauthorized users.

We keep the selection of k points dynamic by the *AS* in each attempt to increase the difficulty of an adversary in correctly guessing the different pieces of the secret $SIMcode_1$. Also, in each such request, n is randomly generated, which is at least $1.05 \times k$. For example, we can divide the hash of secret $SIMcode_1$ into twenty parts (n = 20) of $S\text{-}Actcode_i$, and any fifteen parts (k = 15) can sufficiently reconstruct the original $SIMcode_1$. Note that the construction of $SIMcode_1$ by an adversary is useless, as it cannot derive or extract meaningful information from $SIMcode_1$ and the information sent over the network. Later, in our protocol after verifying sender *MS* and all recipients MS_i, the *AS* sends a new $new\text{-}SIMcode_1$ and $new\text{-}SIMcode_i$ to the *MS* and all MS_i, respectively, for subsequent authentication request.

2) Pseudo-Identity Generation: The generation of *TID* and retrieval of *IMSI* are not publicly available. We consider *IMSI* 128-bit as defined in the *3GPP* specifications [60], according to which the length of the compressed *IMSI* and encrypted *IMSI* shall be 64 bits (8 octets) and 128 bits (16 octets), respectively. We use an encryption function to generate a temporary identity of each participating user. Each MS_i (including sender *MS*) computes TID_i as $TID_i = f_2(IMSI_i, T_i)_{DK_i}$ to prevent the transmission of the original $IMSI_i$ over the network that protects *ID*-theft, eavesdropping, and *MITM* attacks. Here, T_i is the current timestamp, DK_i is a delegation key, and $f_2()$ is a reversible symmetric encryption function (*e.g.*, *AES-CTR*). The structure of this function may be known; however, DK_i key remains secret.

3. Protocol Initialization: Let m be the total number of authentication requests generated by various mobile users MS_i (where i = 2, 3, ..., m+1) to the *AS* at the same time when they receive a request from the sender *MS*. Initially, each MS_i (and sender *MS*) chooses a random number $K_i \in \mathbb{Z}_p^*$ (where p is a large prime integer of 128 bits), generates current timestamp T_i, and derives a delegation key DK_i, where $DK_i = f_1(T_i)_{SK_i}$ and $f_1()$ is a hash-based *MAC* function, such as *HMACSHA-256*. Thereafter, each MS_i computes $Y_i = K_i \oplus IMSI_i$ and a symmetric-signature $Z_i = (K_i + DK_i \oplus ReqNo) \bmod m$, where $\oplus$ is a bitwise *XOR* operation. Each mobile user generates a valid symmetric-signature and fulfills the security properties with Assumption 3, such as *authenticity* (the signer itself signs the associated message with its key), *unforgeability* (only the signer can generate a valid symmetric-signature for the associated message, assuming an honest *AS*), *non-reusability* (generated symmetric-signature cannot be reused), *non-repudiation* (signer cannot deny the signing of a message, *i.e.*, symmetric-signature, with a honest *AS*), and *integrity* (ensures that content has not been modified). Note that in symmetric key cryptography, both parties know the shared secret key. If they send messages to a third party, then it is difficult to determine the sender of the message received by a third party. In such a scenario, only two parties are involved. In other words, only the MS_i (and sender *MS*) and the *AS* know the corresponding SK_i key as well as the generated DK_i key.

4. Protocol Execution: This phase proceeds the execution of the proposed protocol in two parts: a fresh batch authentication and the subsequent authentications. A batch authentication process verifies the identites of the sender and all receiver *MS* simultaneously. If a batch process contains malicious *MS* and does not successfully verifies the identities of all users, a re-batch authentication will be performed after detecting the malicious users in the batch. Once a batch authentication is successful, the verified sender and receiver users can directly communicate with each other using a delegation key with a specified expiry time.

Phase-1: Batch Authentication: The proposed protocol performs the following six steps for a batch authentication:
Step 1. [*MS* $\rightarrow$ MS_i: T_1, *ReqNo*, TID_1, $S\text{-}Actcode_i$, $MAC1_1$]: The sender *MS* sends its request as {timestamp T_1, *ReqNo*, temporary identity TID_1, activation code $S\text{-}Actcode_i$, $MAC1_1$}, where $MAC1_1 = f_3(T_1, ReqNo, TID_1, S\text{-}Actcode_i)$ to all targeted MS_i (message-1), where $f_3()$ is a hash-based *MAC* function, such as *HMAC-SHA1*.
Step 2. [MS_i $\rightarrow$ *AS:* T_1, *ReqNo*, TID_1, $S\text{-}Actcode_i$, $MAC1_1$, $Actcode_i$, TID_i, T_i, Y_i, Z_i, $MAC2_i$]: On receiving the request, all recipients MS_i compute $MAC1_1^{'}$ and verify $MAC1_1 \stackrel{?}{=} MAC1_1^{'}$. If it verifies, then the respective MS_i proceeds; otherwise, the connection is terminated by the MS_i. The MS_i who successfully verify $MAC1_1$, compute and send their activation codes $Actcode_i$, temporary identity TID_i, timestamps T_i, variables Y_i and Z_i, and $MAC2_i$ to the *AS* along with message-1 received from the *MS* except $MAC1_1$ (message-2), where $MAC2_i = f_3(T_1, ReqNo, TID_1, S\text{-}Actcode_i, T_i, TID_i, Y_i, Z_i, Actcode_i)$.
Step 3. [*AS* $\rightarrow$ *MS:* $E\{T_{m+1}, \boldsymbol{ExpT}\}_{DK_1}$]: On receiving the message, the *AS* computes $MAC2_i^{'}$ for all the received messages from different MS_i and compares $MAC2_i \stackrel{?}{=} MAC2_i^{'}$. If the verification returns false, the *AS* terminates the connection for the MS_i. Otherwise, the *AS* extracts the hashed $SIMcode_i$ and computes DK_i key (from the respective $Actcode_i$ and SK_i) and $IMSI_i = f_2(TID_i, T_i)_{DK_i}$ (from the received TID_i) for all valid MS_i. The *AS* also computes the hashed $SIMcode_1$, extracts SK_1 key, derives DK_1 key and retrieves $IMSI_1$. Thereafter, the *AS* computes $P_i = (DK_i \oplus IMSI_i)$, $P = \sum_{i=1}^{m}(P_i)$, $R = \sum_{i=1}^{m}(Z_i \oplus IMSI_i) - (ReqNo \oplus P)$ and $Q = \sum_{i=1}^{m}(Y_i)$. If $(Q \stackrel{?}{=} R)$ is true at the *AS*, all MS_i are successfully verified by the *AS*. Otherwise, one or more MS_i are malicious, which requires a re-batch authentication.

Re-batch Authentication Process: In a re-batch authentication, the *AS* finds all invalid MS_i using a detection algorithm "Malicious_Requests_Detection" (discussed in Section 4) and removes all invalid MS_i from the batch. After removing malicious MS_i from a

batch, the *AS* re-computes $P = \sum_{i=1}^{m-t}(DK_i \oplus IMSI_i)$ and $R = \sum_{i=1}^{m-t}(Z_i \oplus IMSI_i) - (ReqNo \oplus P)$, where t is the total number of malicious MS_i. Thereafter, the *AS* compares $\sum_{i=1}^{m-t}(Y_i \stackrel{?}{=} R)$, and ensures that all legitimate MS_i are authenticated. Finally, the *AS* sends $E\{T_{m+1}, ExpT\}_{DK_1}$ to the sender *MS* (message-3), where *new-ReqNo* is a new request number assigned by the *AS* for subsequent request.

Step 4. [*MS* → *AS:* $E\{T_{m+1}\}_{DK_1}$]: The *MS* replies $E\{T_{m+1}\}_{DK_1}$ as an acknowledgment to the *AS* (message-4).

Step 5. [*AS* → *MS*$_i$: P_i, $E\{T_{m+2}$, ***new-ReqNo*, ***ExpT***, ***new-SIMcode***$_i$, ***DK***$_1\}_{DK_i}$, ***MAC3***$_i$]:** The *AS* decrypts the message as $D\{E\{T_{m+1}\}_{DK_1}\}_{DK_1}$ and verifies T_{m+1}. Furthermore, the *AS* sends all P_i to the respective MS_i along with $\{E\{T_{m+2}, new\text{-}ReqNo, new\text{-}SIMcode_i, ExpT, DK_1\}_{DK_i}, MAC3_i\}$ (message-5), where $MAC3_i = f_3(P_i, E\{T_{m+2}, new\text{-}ReqNo, new\text{-}SIMcode_i, ExpT, DK_1\}_{DK_i})$. On receiving the message, all MS_i compute $MAC3_i^{'}$ and compare $MAC3_i \stackrel{?}{=} MAC3_i^{'}$. If it holds, all MS_i compute $P_i^{'}$ and compare $P_i \stackrel{?}{=} P_i^{'}$, where $P_i^{'} = (DK_i \oplus IMSI_i)$. If the verification returns true, the *AS* is verified by all MS_i. Otherwise, the particular MS_i terminates the connection.

Step 6. [*AS* → *MS:* $E\{T_1$, ***new-ReqNo, new-SIMcode*$_1\}_{DK_1}$]:** Finally, the *AS* sends $E\{T_1, new\text{-}ReqNo, new\text{-}SIMcode_1\}_{DK_1}$ to the *MS* (message-6), where T_1 (first timestamp) shows the completion of authentication process. Thereafter, endpoints can communicate with secure messages encrypted by *AES-CTR* with 128 bits key.

Phase-2: Subsequent Authentications: Any subsequent request made by sender *MS* within a pre-determined expiry time of DK_1 executes as follows:

Step 7. [*MS* → *MS*$_i$: ***new-ReqNo,* $E\{$***TID***$_1, T_j\}_{DK_1}$]:** The *MS* sends $\{new\text{-}ReqNo, E\{TID_1, T_j\}_{DK_1}\}$ to all respective MS_i (message-7).

Step 8. [*MS*$_i$ → *MS:* *E*$\{$*new-ReqNo***$_i$, $T_{j+1}\}_{DK_1}$]:** All MS_i check *new-ReqNo*, retrieve the corresponding DK_1 from their memory, and decrypt the received message. Furthermore, if $T_j \leq ExpT$, all respective MS_i compute another request number $new\text{-}ReqNo_i = f_3(new\text{-}ReqNo, TID_1, T_j)$ and sends it to the *MS* along with T_{j+1} (message-8). The same $new\text{-}ReqNo_i$ is computed by each MS_i. However, T_{j+1} is different for each MS_i. Thereafter, the *MS* retrieves the message and stores $new\text{-}ReqNo_i$ in its memory for the subsequent request.

4 Discussion

In this section, we discuss the reliability of the *BVPSMS* protocol, an algorithm to detect malicious requests, and the impact of user mobility [14], [48].

Proposition 1. *Hypergeometric distribution probability helps to predict malicious requests in a batch and determines the reliability of the proposed protocol.*

If we can determine the approximate number of malicious user requests involved in the process, Hypergeometric distribution probability can help us determining the probability in detecting malicious requests in our system. Deploying an Intrusion Detection System (*IDS*), such as the one presented in [61] in the cellular network, can identify the suspicious malicious users. Let N_{MS} be the maximum number of authentication requests generated by mobile users

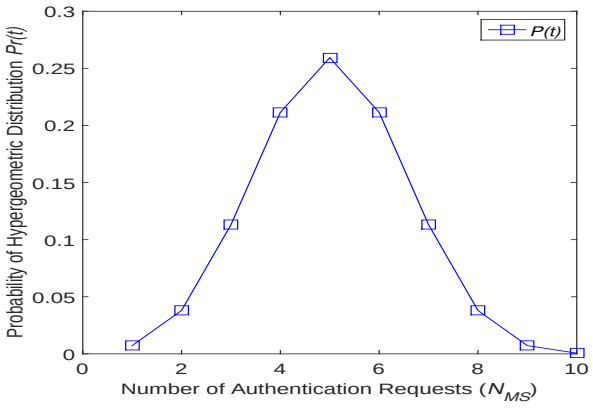


Fig. 4: Reliability analysis of our protocol when t = [1-10].

at any point of time. Realistically, some of these requests may be malicious, denoted as N_{IN}. Also, we assume that N_{AS} is the maximum capacity of the *AS* to authenticate requests at any point of time. For the statistical analysis, we assume that N_{MS} = 100, N_{AS} = 50, and N_{IN} = 10% of the N_{MS}, *i.e.*, 10. Let $Prob\{t\}$ is the probability when t malicious authentication requests are sent to the *AS*. The probability of Hypergeometric distribution [62] is as follows:

$$Prob\{t\} = \frac{\binom{N_{MS}-N_{IN}}{N_{AS}-t}\binom{N_{IN}}{t}}{\binom{N_{MS}}{N_{AS}}}, where\ t = 1, 2, ..., 10.$$

This indicates that $(N_{AS} - t)$ valid requests are sent out of $(N_{MS} - N_{IN})$. Figure 4 shows the probability of Hypergeometric distribution when N_{MS} = 100, N_{AS} = 50, N_{IN} = 10, and malicious requests are t = 1, 2, 3, ..., 10. This probability is maximum (0.25) for t = 5 (half of t), and minimum (0.00059) for t = 10 (last of t values).

Proposition 2. *There exists an algorithm that detects malicious requests in a batch.*

In practice, few of the mobile participants may be dishonest or malicious. A dishonest mobile participant will always lie about the true secret value. Our scheme assumes that all shares lie on a single polynomial of degree at most k-1. This might not hold if the sender mobile user is dishonest

Algorithm 1 *Malicious_Requests_Detection*

Input: The *AS* receives a set (*AR*) of m-authentication requests (R_i) as $AR = \{R_1, R_2, R_3, ..., R_m\}$ at any time.
Output: Returns a set of malicious requests (*MR*), otherwise returns True.

if (batch verification (*AR*, m) == 1) **then** returns True.
else
  while (batch verification (*AR*, m) != 1) **do**
    $AR_1 = \{R_1, R_2, R_3, ..., R_{\lceil m/2 \rceil}\}$;
    $AR_2 = \{R_{\lceil m/2 \rceil+1}, R_{\lceil m/2 \rceil+2}, R_{\lceil m/2 \rceil+3}, ..., R_m\}$;
    batch verification (AR_1, $\lceil m/2 \rceil$);
    batch verification (AR_2, $m - \lceil m/2 \rceil$);
    if (m == 1 && batch verification (*AR*, m) != 1) **then** returns $MR = \{IMSI_i\}$

or malicious and sends bad shares to some of the mobile recipients. However, our system model has a honest sender mobile user. But a mobile user participant who lies about his share can cause reconstructing incorrect value of the secret (hash of *SIMcode*) at the server. Our scheme is a fault-tolerant scheme that allows the hash of *SIMcode* to be correctly reconstructed, even in the presence of a certain number of corrupted shares.

We propose an algorithm to detect malicious requests of the MS_i in a batch in at most $log\ m$ verification rounds (O($log\ m$)). The proposed algorithm, based on binary search approach, is explained as Algorithm 1. Only the hash-based search complexity is better than binary search. The hash-based searching is useful when you know the data, and even more efficient when the data is in sorted order (O(1)). However, in our protocol, the *AS* neither knows the actual data nor stores any data until it is verified. In such case, the proposed algorithm for malicious detection is suitable. Note that "batch verification (*AR*, m)" is the batch verification process at the *AS* involving P, R, and Y_i as explained in our protocol. Each invalid MS_i is placed on a black-list and can only be removed once the predefined time is over. During this period, the request from particular MS_i is discarded.

More generally, if there can be t malicious users with faked shares (*S-Actcode*$_i^{'}$, $i = 1, 2, ..., t$), we can show that the secret can be recovered and the malicious users identified if $k + 2t$ shares are available for reconstruction. In other words, we need at least $k + t$ honest shares available (in addition to the t possible malicious users) in order to recover the secret (hash of *SIMcode*) and identify the malicious users. We assume that there are t cheaters or malicious users participating at any time, where $t \leq k/2$. In any secret sharing cheater or malicious identification scheme, the optimal cheating threshold is $k = 2t + 1$. In [63], it is shown that in any such scheme, the following lower bound must be satisfied: $|V| \geq (|S - Actcode| - 1)/\epsilon + 1$, where $|V|$ exactly matches the above bound is said to be optimal. Let $k = 2t+1$, $p = 1/\epsilon$ and $|S - Actcode| = p^i$, where $i > 1$ and $S{-}Actcode = (S{-}Actcode_1, S{-}Actcode_2, ..., S{-}Actcode_i)$ is a shared secret. We can identify up to t malicious users such that $|V| = |S - Actcode|/\epsilon^{3n}$ [64]. Now, we assume that j $(n \geq j \geq t)$ number of participants are involved in a secret reconstruction out of n. Then, we have $j - t$ legitimate shares in a secret reconstruction. When $j - t > t$ $(j \geq t + 1)$, there are $\binom{j-t}{t}$ cases that will construct the legitimate secret [65]. This attack of not being able to reconstruct the secret succeeds only when $j - t < t$.

Proposition 3. *There is a sustainable impact of mobility when a user moves out of range of the home AS.*

One of the challenges of the cellular networks is to provide a reliable and secure service to a mobile user when he/she moves to a roaming area. The proposed protocol works only if the *AS* receives at least k messages from the participated users. It is also assumed that the *AS*s are deployed at different geographic locations similar to traditional cellular networks, and are interconnected to each other with a pre-shared secret key between each pair of the *AS*s. When a roaming mobile user requests for an *SMS* service, the corresponding *AS* of that area handles the request, and sends the request message encrypted with pre-shared key to the home *AS* of the user. The protocol execution takes place at the home *AS* and the result is returned securely to the roaming *AS* securely. Finally, the roaming *AS* grants/revokes *SMS* service to the respective mobile user. Also, if one or more *MS*s are out of network, the *AS* will verify whether it has received at least k messages from different *MS*s. If it holds, the *AS* proceeds, otherwise the *AS* waits for a timeout period. If the *AS* still does not receive k messages, it discards the connections, and notifies the sender *MS* to restart its request.

5 Security Analysis

This section describes provable security of the proposed protocol by achieves the security goals outlined in Section 2.2 along with a formal verification of the proposed protocol. Provable security aims to give an guarantee that the proposed protocol is secure under a system model (described in Section 2.1) and cannot be broken by a class of adversaries under a threat model (described in Section 2.2).

The proposed protocol will not affect network and service control compliances with the relevant regulation in different countries with respect to spam and marketing. A sender can only send an *SMS* to a receiver if the receiver already agreed to participate and the identities of both users are verified by the authentication server. In most of these services, the recipient knows the actual mobile number (individual or service number) of the sender. The proposed protocol hides the actual identity of each user over the network. Hence, it is unlikely that spam messages could be sent using this protocol. This section further analyses different security properties fulfilled by the proposed protocol.

Property 1. Mutual Authentication. *The proposed protocol provides mutual authentication between all MS/MS_i and the AS.*

The *BVPSMS* protocol provides mutual authentication between the *AS* and the *MS*, and between the *AS* and the MS_i. The *AS* authenticates all MS_i by verifying $\sum_{i=1}^{m}(Y_i \stackrel{?}{=} R)$ while each MS_i authenticates the *AS* by comparing $P_i \stackrel{?}{=} P_i^{'}$. The sender *MS* authenticates the *AS* by decrypting the received message-3 using DK_1 while the *MS* is authenticated by the *AS* by verifying T_{m+1}.
$MS \mid\equiv MS_i \mid\equiv AS \rightarrow MS \mid\equiv MS_i \wedge MS \mid\equiv AS.$

Property 2. Secure Session Keys. *The protocol initiates a secure session key establishment between all MS/MS_i and the AS. In fact, Adversary $\mathcal{A}$ will not be successful in obtaining SK_1/SK_i or DK_1/DK_i key, even if it captures S-Actcode$_i$/Actcode$_i$ of a MS.*

A unique DK_i key is used within the expiry of a session for each authentication between the *AS* and each MS_i. $\mathcal{A}$ is unable to generate DK_1/DK_i key as it does not know the SK_i key and the key generation function $f_1()$. Since each *S-Actcode$_i$/Actcode$_i$* is sent over the network only once, the protocol is secure even if $\mathcal{A}$ is able to capture *S-Actcode$_i$/Actcode$_i$*. Moreover, $\mathcal{A}$ cannot derive any relation among captured *S-Actcode$_i$/Actcode$_i$*, as *SIMcode$_1$/SIMcode$_i$* are randomly generated at the *AS*. Moreover, after each authentication, *new-SIMcode$_1$/new-SIMcode$_i$* are sent to each involved MS/MS_i. If $\mathcal{A}$ modifies *Actcode$_i$* in message-2, the

computed $MAC2_i^{'}$ will not match with the received $MAC2_i$ at the *AS*. Hence, the MS_i will terminate the connection.

Property 3. User Privacy. *Adversary $\mathcal{A}$ cannot trace the original identity of the MS/MS_i. In fact, $\mathcal{A}$ is not able to identify the actual user, even if it captures the TID_1/TID_i of a mobile user.*

If adversary $\mathcal{A}$ is able to retrieve the original identity (IMSI) of a user, it can perform an impersonation and session hijacking attacks [66]. Our protocol preserves identity anonymity and untraceability properties. The user's anonymity and untraceability guarantee that besides the user and the *AS* $\in$\{networks\}, no one including the operator: (i) can retrieve the actual identity of the user, and (ii) is able to identify previous sessions involving that user.

Untraceability: Our protocol satisfies untraceability as $\mathcal{A}$ cannot distinguish whether two *TID*s correspond to the same *MS*/MS_i or two different *MS*/MS_i.

$Verify(publicChannel)[(IMSI_1, IMSI_2)|TID_i|MS/MS_i|AS] \approx Verify(publicChannel)[IMSI_1|IMSI_2|TID_i|MS/MS_i|AS]$.

In our protocol, privacy of each MS_i (including *MS*) is ensured. Each TID_i is computed from the original $IMSI_i$ as $TID_i = f_2(IMSI_i, T_i)_{DK_i}$ before a message is sent by each MS_i over the network. We implement $f_2()$ using *AES-CTR* with DK_i key since no practical full attack has revealed against on *AES*. As TID_i is used by each MS_i over the network, $\mathcal{A}$ is unable to trace the original identity of the user.

Indistinguishability under Anonymous Identity: Our protocol is indistinguishable under anonymous identity as no adversary $\mathcal{A}$ at time t can distinguish between two chosen identities TID_1 and TID_2 with a negligible ϵ advantage.

$Pr[\mathcal{A}(TID_1) = 1] - Pr[\mathcal{A}(TID_2) = 1] \leq \epsilon$.

$\mathcal{A}$ cannot distinguish and relate TID_i and other messages with ID_i, as each TID_i is used only once over the network. For all subsequent requests, a different *new-ReqNo*$_i$ is used each time when the sender *MS* connects to the MS_i. The MS_i sends an encrypted *new-ReqNo*$_i$ to the *MS* that will be used for the next authentication within a session. Hence, untraceability and identity anonymity are ensured, as $\mathcal{A}$ cannot trace TID_i, *SIMcode*$_i$, and *new-ReqNo* to link with users, and also $IMSI_i$ would not be revealed to $\mathcal{A}$ and intermediate operators.

Property 4. Defeating Linkability and IND-CPA Attacks. *$\mathcal{A}$ cannot link current session information with previous sessions. Our protocol maintains perfect forward secrecy and Indistinguishability under Chosen Plaintext Attack (IND-CPA). The protocol achieves fairness and guarantees that "no MS_i (malicious or legitimate) has an advantage" and maintains correctness under honest, semi-honest, and dishonest majority scenarios.*

The MS_i (including *MS*) and the *AS* generate fresh DK_i keys with unique timestamps, TID_i, *Actcode*$_i$ and K_i. Therefore, $\mathcal{A}$ cannot retrieve the information based on linkability.

Forward Secrecy: Our protocol maintains forward secrecy as no $\mathcal{A}$ could obtain past keys and generate future keys.

The SK_i and DK_i keys are never sent over the network, and a new DK_i key is used in each fresh session to encrypt $IMSI_i$ using *AES-CTR*. Even compromising current DK_i will not allow $\mathcal{A}$ to obtain or generate past and future keys. Also, the past keys cannot be used for future sessions, as endpoints generate a fresh DK_i key.

IND-CPA: Our protocol is *IND-CPA* secure as no adversary $\mathcal{A}$ in time t can distinguish between two chosen messages msg_1 and msg_2, and has no or negligible advantage. $\underset{DK_i \leftarrow SK_i}{Pr}[\mathcal{A}(msg_1) = 1] - \underset{DK_i \leftarrow SK_i}{Pr}[\mathcal{A}(msg_2) = 1] \leq \epsilon$.

Assuming that $\mathcal{A}$ has unlimited access to the encrypted data using a random oracle, the messages encrypted by the same key in our protocol generate different ciphertexts. Even encrypting the same plaintext with the same key generates different ciphertext, as at least one of the input parameters of the message is always different. The MS_i generates TID_i as $f_2(IMSI_i, T_i)_{DK_i}$, where T_i changes for each fresh message. We use *AES-CTR* as $f_2()$ that encrypts successive values of a counter with *AES*, and regurgitates concatenation of the encrypted blocks. *AES-CTR* stream never includes twice the same block and is *IND-CPA*.

A protocol is said to be fair if it ensures that no user can gain a significant advantage over other users, even if the protocol halts for any reason. In our protocol, the *MS*/MS_i and the *AS* learn each others' information. However, the *MS* and the MS_i cannot learn any information about each other, as one user is unable to obtain DK_i keys belonging to other users. Users are also unable to derive $IMSI_i/TID_i$ of each others, as each DK_i is secret. Also, $\mathcal{A}$ cannot generate a valid symmetric-signature S_i, as it does not know the correct SK_i and/or DK_i keys, and K_i is randomly generated by each MS_i. Our protocol also maintains *IND-CPA*; therefore, no MS_i has an advantage over others. The proposed protocol fairly works under all three scenarios. We consider these scenarios only for the MS_i, not for the *AS*. The reason is that the *AS* keeps SK_i keys of all MS_i secret. Hence, it cannot be dishonest or semi-dishonest. The effectiveness of our protocol under all three scenarios can be observed by re-batch verification time discussed in Section 6.2.3. Our protocol maintains security properties under these scenarios, such as *IND-CPA*, forward secrecy and fairness.

Property 5. Defeating Other Security Attacks. *Our protocol defeats SMS disclosure, SMS spoofing, replay, MITM, and impersonation attacks between the MS/MS_i and the AS. The protocol provides security protection over-the-air and SS7 channel. $\mathcal{A}$ cannot compromise message confidentiality and integrity. The protocol is secure against both passive and active corruption attacks in the presence of non-adaptive and/or adaptive adversaries.*

BVPSMS provides mutual authentication between the *AS* and the *MS*/MS_i by verifying $(\sum_{i=1}^{m} X_i) \stackrel{?}{=} R$, and $P_i \stackrel{?}{=} P_i^{'}$. It prevents the system against impersonation attack. Transmitted messages are securely encrypted using *AES-CTR*, which protects the system against *SMS* disclosure and *MITM* attacks. $\mathcal{A}$ is unable capture actual *IMSI* using *IMSI* catcher as each *MS*/MS_i sends its *TID* over the network. It also prevents *SMS* spoofing as $\mathcal{A}$ cannot impersonate the server as a legitimate user because *TID* changes each time a user communicates. Furthermore, a timestamp value sent with each message protects the system against replay attack. Our protocol provides end-to-end *SMS* security from the *MS* to all MS_i over *OTA* interface and *SS7* channel, as each confidential message is encrypted using *AES-CTR*

TABLE 2: Protocols Comparison

Prevention Goals	*ABAKA* [14]	*RAISE* [12]	*SPECS* [17]	*b-SPECS+* [15]	*BVPSMS*
Mutual Authentication	Yes	No	Yes	Yes	Yes
User Privacy	Yes	Yes	No	No	Yes
Integrity Protection	No	Yes	No	No	Yes
Replay Attack	Yes	Yes	No	No	Yes
MITM Attack	Yes	Yes	Yes	Yes	Yes
Impersonation Attack	Yes	Partial	No	Yes	Yes

with a 128-bit key. Moreover, message integrity (message content and its threshold delivery in time) is maintained, as $T_{receive} \leq T_{generate} + T_{threshold}$ and *MAC*s are used for verification and the messages received after will be lapsed.

In passive and active corruption attacks, $\mathcal{A}$ obtains complete information held by the corrupted MS_i (while a MS_i still runs protocol correctly) and $\mathcal{A}$ takes over control of corrupted MS_i, respectively. In both cases, our protocol maintains *IND-CPA* indistinguishability as well as perfect forward secrecy. Moreover, keys are never sent over the network, and delegation keys are generated only for a session. Furthermore, both passive and active adversaries can be non-adaptive (a set of corrupted MS_i is chosen before the protocol starts) or adaptive (a corrupted MS_i is selected at any time during protocol run). In any case, $\mathcal{A}$ acting as corrupted MS_i does not affect the security of the protocol.

Table 2 lists the security and privacy requirements achieved by the existing protocols. These protocols are secure against *MITM* attack, but do not provide integrity protection to the messages with the exception of *RAISE* [12]. However, *RAISE* [12] does not provide mutual authentication and is only partially secure against impersonation attacks. We observe that user privacy is preserved in *ABAKA* [14] and *RAISE* [12], but both *SPECS* [17] and *b-SPECS+* [15] suffer from replay attack. Our proposed protocol, on the other hand, fulfills all the mentioned requirements.

6 Performance Evaluation

This section presents the performance evaluation of *BVPSMS* in terms of overheads, verification and re-batch verification times, and the time, space, and cost analysis.

6.1 Analysis

This subsection analyzes the performance of the *BVPSMS* protocol. We compare the communication overhead generated by *RAISE* [12], *ABAKA* [14], *SPECS* [17], and *b-SPECS+* [15] along with the *BVPSMS* protocol. There is no batch protocol for *SMS* security in the literature. However, we compare the communication overhead generated by the protocols with our protocol, as all protocols are based on authentication considering the same wireless network communication scenario, and also the flow of information is same in all the protocols. However, the computation overhead and verification delay are different in both types of the protocols because *VANET* protocols have additional devices and road side equipment to communicate information over the network.

6.1.1 Communication Overhead

Let m be the number of recipients MS_i, and r be the number of subsequent multiple authentication requests within the expiry time, *i.e.*, *ExpT*. The communication overhead can be defined as the total number of bits transmitted during the authentication process over the network. The transmission overhead generated by the *BVPSMS* protocol during m-authentication requests can be evaluated as:
Phase-1: Total number of transmitted bits = (1)+(2)+(3)+(4)+(5)+(6) = (128+64+8+64+64)$\times m$ + (128+64+8+64+64+128+64+128+128+64+64)$\times m$ + (64+64+8) + (64) + (128+64+64+8+64+64+128)$\times m$ + (8+64+64) = 336+1752$\times m$ bits.
Phase-2: Total number of transmitted bits = ((7)+(8))$\times r$ = (128+8+64)$\times r$ + (64+8)$\times r$ = 200$\times r$.
Total overhead = 42+(219$\times m$)+(25$\times r$) bytes.

BVPSMS is our original protocol that provides integrity to each message in two phases. Since all the protocols except *RAISE* [12] compared in Table 3 provide no integrity, we use two variants of *BVPSMS* for comparison: *BVPSMS** for fresh authentication without integrity protection (as phase-1), and *BVPSMS*** for each subsequent authentication within the expiry time of DK_1 key (as phase-2). For m-authentication requests, *BVPSMS** generates 154$\times m$ bytes overhead, which is lowest among all the protocols discussed in the paper, while for all subsequent authentication requests, the overhead is only 34$\times r$ bytes.

From Figure 5, it is clear that *BVPSMS** and *BVPSMS*** generate less communication overhead among all protocols. *BVPSMS** reduces the communication overhead by 6.1%, 23%, 12.5%, and 46.52% in comparison to *ABAKA*, *RAISE*, *SPECS*, and *b-SPECS+*, respectively, when m = 5, 10, 20, 50, 100. For any subsequent authentication request, *BVPSMS*** produces significantly low overhead in comparison to all the protocols. It reduces the communication overhead by 79.27%, 89.83%, 80.69%, and 88.2% in comparison to *ABAKA*, *RAISE*, *SPECS*, and *b-SPECS+*, respectively, when r = 5, 10, 20, 50, 100.

In the proposed protocol, the receiver *MS* will not be liable for any cost. Only the sender *MS* will pay the cost for sending the *SMS*s. Technically, there is no limitation of sending or receiving *SMS*s in the modern cellular infrastructure, assuming the upload and download data speeds

TABLE 3: Communication Overhead in Batch Authentication by Different Protocols

Protocols	Device-Server (bytes)	Intermediate Authority Server (bytes)	Server-Device (bytes)	Total (bytes)
ABAKA [14]	84$\times m$	–	80$\times m$	164$\times m$
RAISE [12]	200$\times m$	–	–	200$\times m$
SPECS [17]	48$\times m$	96$\times m$	32$\times m$	176$\times m$
b-SPECS+ [15]	48$\times m$	176$\times m$	64$\times m$	288$\times m$
BVPSMS*	97$\times m$	–	57$\times m$	154$\times m$
BVPSMS**	25$\times r$	9$\times r$	–	34$\times r$

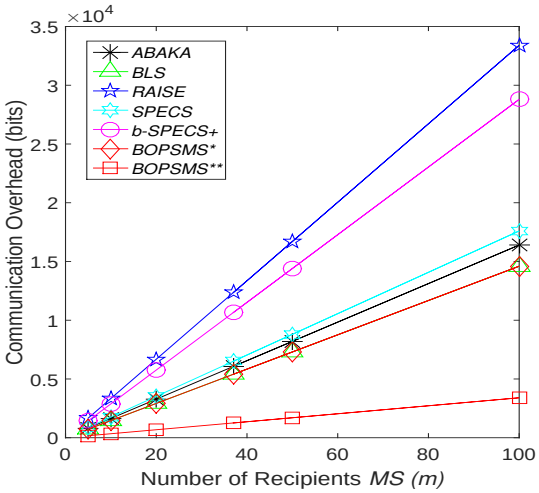


Fig. 5: Communication overhead analysis.

are sufficiently fast. For example, 4*G LTE* offers typical download and upload speeds of 14Mbps and 8Mbps, respectively, whereas 4*G LTE*-Advanced provides typical real world download speeds of 42Mbps and upload speeds of 30Mbps [67]. At the time of this research, voice over *LTE* provides the fastest platform for sending an *SMS* [68]. The proposed protocol is efficient and practical as long as the authentication server is able to handle a batch process efficiently. The delivery of each *SMS* is handled by the *SMS* center, which keeps any undelivered *SMS* for 24 hours. Practically, it can handle a very large number of undelivered *SMS*s in the system.

6.1.2 Computation Overhead

The computation overhead generated by *BVPSMS* during m-authentication requests is shown in Table 4. We consider all functions as a single unit cost. Then, the computations at the *MS*, MS_i, and *AS* are as follows:

Phase-1: At MS = 8, MS_i = 11×m, and AS = 6+14×m.

Phase-2: At MS = 2×r and MS_i = 2×r.

Total computation overhead = 8 + (11×m) + (6+14×m) + (2×r) + (2×r) = 14+(25×m)+(4×r) bits.

We compute the communication and computation overheads (in bits) generated by our protocol when m = 10, 20, 50, 100; r = 1, 2, 5, 10. For m=100, the generated communication overheads are 2745.875 bytes and 2970.875 bytes, respectively, when r=1 and r=10. Similarly, when m=100, the computation overheads for r=1 and r=10 are 314.75 bytes and 319.25 bytes, respectively. The computation overhead of the proposed protocol is much lower than the communication overhead, therefore, the proposed protocol achieves lightweight computations and can support a large number of mobile users to be authenticated in a batch. This also indicates that our protocol is efficient even when a large number of subsequent requests is executed.

TABLE 4: Computation Overhead in Batch Authentication

Entity Name	Total Computation (Time Computation)
Phase-1	
At the *MS*	$T_{f_1()}$, $T_{f_2()}$, $2T_{D\{\}_{DK_1}}$, $T_{E\{\}_{DK}}$, $T_{f_3()}$, $T_{H()}$, $T_{S\text{-}Actcode}$
At the MS_i	$mT_{f_1()}$, $mT_{f_2()}$, $3mT_{XOR}$, mT_{Add}, $mT_{D\{\}_{DK}}$, $2mT_{f_3()}$, $mT_{Actcode}$, $mT_{H()}$
At the *AS*	$(m+1)T_{f_1()}$, $(m+1)T_{f_2()}$, $(m+1)T_{H()}$, $3mT_{f_3()}$, $(3m-3)T_{Add}$, T_{Sub}, $(2m+1)T_{XOR}$, $(m+2)T_{E\{\}_{DK}}$, $T_{D\{\}_{DK}}$, $(m+1)T_{SIMcode}$
Phase-2	
At the *MS*	$rT_{E\{\}_{DK}}$, $rT_{D\{\}_{DK}}$
At the MS_i	$rT_{D\{\}_{DK}}$, $rT_{E\{\}_{DK}}$

6.2 Simulation

This section presents the simulation results of our protocol in terms of the total execution and verification times. We also perform time, space, and cost analysis of our protocol.

6.2.1 Protocol Execution Time

We implemented a client-server paradigm for our system, where the MS/MS_i are the clients and the *AS* is a server. We performed various operations on an Intel Core i3-2330M 2.20GHz machine with Windows7 OS, 256 MB RAM, using JDK1.7 with J2ME WTK mobile emulator. On average, the execution time to perform addition, *XOR*, and subtraction are T_{add} = 0.0009 milliseconds (ms), T_{xor} = 0.03 ms, and T_{sub} = 0.0009 ms, respectively. We setup the system with 50 MS_i (and one *MS*) transmitting their messages to the server *AS*, when the *MS* sends an *SMS* to these MS_i. The average value of 30 iterations is considered for each result.

Note that protocol execution time is the complete time for mutual authentication between all MS/MS_i and the *AS*. Table 5 shows our simulation results obtained for various functions' computations. Here, *Ext*, *TUM*, *Enc*, and *Dec* are the execution time (ms), total used memory (bytes), encryption, and decryption process, respectively. The $f_2()$ is implemented as *AES-CTR*, where encryption (generation of TID_i) took 13.6 ms and decryption (generation of $IMSI_i$) is performed in 4.2 ms. The *AES* is a secure algorithm till date and *CTR* mode provide parallelism to speed up the execution. The same results are obtained for $E\{\}_{DK_1}/D\{\}_{DK_1}$ using *AES-CTR*. The $f_1()$ and $f_3()$ are implemented as *HMACSHA-256* and *HMAC-SHA1*, respectively. The output of *HMAC-SHA1* and *HMACSHA-256* are truncated to 64 and 128 bits, respectively because the output of $f_3()$ is 64 bits *MAC*, whereas the output of $f_1()$ is 128 bits, which is DK_i key. The input to the *HAMCSHA1* and *HMACSHA-256* are 512 bits each (actual input size plus trailing zeros to make it multiple of 512). Also, the execution time of *SIMcode* using a random number generation and hash generation time of $H()$ using *SHA256* are 0.89 ms and 20 ms, respectively.

TABLE 5: Computations of Various Used Functions

Function	*ExT* (ms)	*TUM* (bytes)
$f_1()$=*HMACSHA-256*	185	15204024
$E\{\}_{DK_1}/f_2()$=*AES-CTR*	13.6	9139681
$D\{\}_{DK_1}/f_2()$=*AES-CTR*	4.2	9124165
$f_3()$=*HMAC-SHA1*	172	15211840
$H()$=*SHA256*	20	14321156

Total execution time of a single authentication:
Phase-1: Total time = transmission time for all messages in phase-1 + time at the entities (*MS*, MS_i, *AS*) = 2.98 sec. Hence, on average the execution time per user = 1.49 sec.
Phase-2: Total time = transmission time for all messages in phase-2 + time at the *MS*, MS_i, *AS* = 10.7+35.6 = 46.3 ms.
Total execution time of a batch authentication:
Phase-1: Total time = transmission time for all messages in phase-1 + time at the *MS*, MS_i, *AS* = 672.20+$m\times$1330.41 ms.
Phase-2: Total time = transmission time for all messages in phase-2 + time at the *MS*, MS_i, *AS* = $r\times$46.3 ms.

6.2.2 Verification Time

The verification delay in our protocol is evaluated between the MS/MS_i and the *AS*. It is the time estimation between the sent messages and the received response or the protocol completion.

BVPSMS Phase-1 *(Time to verify):* MS_i *by AS* = 0.0282+391.55$\times m$ ms, *MS by AS* = 236.40+172.89$\times m$ ms, *AS by* MS_i = 172.03$\times m$ ms, and *AS by MS* 4.2 ms.
Total delay in phase-1 = 240.62+736.47$\times m$ ms.

BVPSMS Phase-2 *(Time to verify): MS by* MS_i = 17.8$\times r$ ms, and MS_i *by MS* 4.2$\times r$ ms.
Total verification delay in phase-2 = 32$\times r$ ms.
Therefore, total verification delay in *BVPSMS* = 240.62+736.47$\times m$+32$\times r$ ms.

6.2.3 Re-batch Verification Time

If a batch authentication is not successful, it is expected to execute a re-batch authentication without including the malicious MS_i. After detecting the malicious MS_i, it is required to remove them from the batch and execute a re-batch authentication process. The delay in re-batch verification can be estimated as follows:
Total delay in a re-batch verification = 0.000933$\times$3$(m-1-t)$ + 0.030322 + 0.000933 = 0.028456+0.002799$\times(m-t)$ ms.

6.2.4 Simulation Results

The execution time of the *BVPSMS* protocol is observed when m = 10, 20, 50, 100; r = 1, 2, 5, 10. For m=100, the protocol execution times are 133.75 sec. and 134.17 sec., respectively, when r=1 and r=10, which are actually on average, 1.32 sec. and 1.21 sec. per user, respectively. It is clear that on average, the execution time per mobile user decreases when r increases. The execution time per mobile user also decreases when m increases and r is fixed. On average, the execution times of our protocol are 1.44, 1.38, 1.35, and 1.34 sec., respectively, when r=10 (fix) and m=10, 20, 50, and 100. The verification times for phase-1 and phase-2 of our protocol are also evaluated when m = 10, 20, 50, 100 and r = 1, 2, 5, 10, 50. For m=100, on average the verification time per user for batch authentication is 0.71 sec. Furthermore, for r=10 and r=50, the total verification times are 0.3 sec. and 1.6 sec., respectively, and on average, the verification time for each subsequent authentication per user is 0.03 sec. It is also clear that the increase in r lowers verification time, on average per mobile user. Re-batch verification time is also computed in our protocol when m = 10, 20, 50, 100 and malicious requests t = 2, 4, 6, 8, 10. For m=10, the re-batch verification times are 0.044 ms, 0.03 ms, and 0.028 ms, respectively, when t=2, t=9, and t=10. Similarly, for m=100, the times are 0.22 ms, 0.21 ms, and 0.20 ms, respectively, when t=2, t=9, and t=10.

6.3 Time, Space, and Cost Analysis

In both single and batch authentications, two functions $f_3()$ and $f_1()$ are implemented as *HMAC* functions. The output of *HMAC-SHA1* and *HMACSHA-256* are 160 bits and 256 bits, respectively. The *DK* key requires 128 from 256 bits and a *MAC* needs 64 out of 160 bits. In total, 192 bits are required to be stored. Furthermore, the time complexity of add, subtract, and *XOR* operations are constant, *i.e.*, O(1). The costs for a single authentication (8 operations) and a batch authentication ($9\times m-1$ operations) are also O(1). The time to compute *Actcode/SIMcode* is constant, and total cost is O(1). The block cipher algorithm, such as *AES*, works with a fixed input size and has O(1) constant complexity. However, when the algorithm has variable length of input (say $|m|$), the time is O(m). The block size is still fixed (128 bits) as the $f_2()$ and $E/D\{\}_{DK_1}$ are implemented using *AES-CTR*. Therefore, the time complexity is independent of input and is constant O(1). Hence, the costs are O(1) for $f_2()$ and $E/D\{\}_{DK_1}$ in a single authentication (2 operations) as well as batch authentication (2$\times m$ operations). The $IMSI_i$ and TID_i of 128 bits each also need to be stored in the memory. Furthermore, the storage is also required for *HMAC-SHA1*, *HMACSHA-256*, and *AES-CTR* at the MS/MS_i as well as at the *AS*. For a re-batch verification, O(1) is only the extra cost need to be paid (for 3$\times m$-3$\times t$+2 operations). Therefore, the *BVPSMS* protocol is an efficient, secure, and cost effective protocol that requires less storage.

7 Conclusion

We proposed a batch verification protocol *BVPSMS* for transmitting secure *SMS* from one *MS* to multiple *MS* recipients. This protocol enjoys several advantages over the related protocols studied in the paper. *BVPSMS* provides mutual authentication between each *MS* and the *AS*. The *AS* efficiently verifies multiple authentication requests sent by different *MS*s at any one time while keeping the original *IMSI* secret during the authentication. We then demonstrated that the protocol is secure against replay attacks, *MITM* attacks, impersonation attacks, *SMS* disclosure and *SMS* spoofing, and also maintains untraceability, forward secrecy, and identity anonymity. The performance results show that in different scenarios, *i.e.*, *BVPSMS*$*$ and *BVPSMS*** when no provision of integrity protection, our protocol incurs a lower communication overhead compared to the protocols studied in this paper. Our evaluation of the protocol using Java demonstrated that the estimated re-batch verification time is almost negligible. The execution and verification times also suggested that our protocol is practical for deployment in real-world cellular networks.

APPENDIX A
FORMAL VERIFICATION

This section presents the formal proof of the proposed scheme using *Proverif*. *Proverif* is an online automated tool to verify whether the logical expressions and the protocol properties are correct and valid with different queries. We perform five $\mathcal{A}$ queries: (i) Can $\mathcal{A}$ successfully recover confidential and useful information from the messages sent over the network?, (ii) Can $\mathcal{A}$ successfully compute parameters generated by the MS?, (iii) Can $\mathcal{A}$ successfully compute parameters generated by the AS?, (iv) Can $\mathcal{A}$ successfully generate DK key of the *MS*?, and (v) Can $\mathcal{A}$ successfully recover secret key of the *MS*?. Following is the syntax and output observed from the *Proverif* tool:

$Neetesh@Neetesh - PC\ /proverif 1.88$
$\$./proverif\ proofs/sms/BVPSMS.pv$

```
free PubChannel: channel.
type host. type nonce. type skey. type sskey. type ident.
(* 3 honest host names MS, MSi and AS *)
free MS, MSi, AS : host.
(* Shared key encryption *)
fun sencrypt(bitstring,sskey): bitstring.
reduc forall x:bitstring, y:sskey; sdecrypt(sencrypt(x,y),y) = x.
fun sencrypttime(bitstring,sskey): nonce.
reduc forall x:nonce, y:sskey; sdecrypttime(sencrypt(x,y),y) = x.
(* Secrecy assumptions *)
not attacker(new DKms1). not attacker(new DKmsi).
(* the table host names/keys *)
table keys(host, sskey).
(* Hash function *)
fun H(bitstring):bitstring.
(* MAC functions *)
fun mac1f3(nonce,bitstring,ident,bitstring): mac.
fun mac2f3(nonce,bitstring,ident,bitstring,nonce,ident,
bitstring,bitstring,bitstring): mac.
fun mac3f3(nonce,nonce,bitstring,bitstring,bitstring,
bitstring): mac.
(* Other function *)
fun f(bitstring): bitstring.
fun f1(nonce,skey): sskey.
fun f2(ident,nonce,sskey): ident.
fun f7(bitstring,bitstring): bitstring.
fun f8(bitstring,sskey,bitstring): bitstring.
fun f9(bitstring,bitstring): bitstring.
fun f10(bitstring,bitstring): bitstring.
fun f11(bitstring,ident): bitstring.
fun f12(sskey,ident): bitstring.
fun f13(bitstring,ident): bitstring.
fun f21(bitstring,bitstring,bitstring): bitstring.
fun f22(bitstring,bitstring): bitstring.
(* Queries *)
event beginMSi(host, host).
event endMSi(host, host).
event beginMS(host, host).
event endMS(host, host).
event beginMSfull(host, host, nonce).
event endMSfull(host, host, nonce).
query x: host, y: host; inj-event(endMS(x,y)) ==>
inj-event(beginMS(x,y)).
query x: host, y: host, z: nonce; inj-event(endMSfull(x,y,z)) ==> inj-
event(beginMSfull(x,y,z)).
(* MS *)
let processMS =
in(PubChannel, (xA1: host, xB1: host));
if xA1 = MS  xB1 = MS then
new imsims1:ident; new skms1:skey; get keys(=MS, skms1) in
let DKms1:sskey = f1(tims1,skms1) in
insert keys(MS,DKms1);
new tims1:nonce; new simcode1:bitstring;
let tidms1:ident = f2(imsims1,tims1,DKms1) in
new rno:bitstring; new q:bitstring; new i:bitstring;
let fofi:bitstring = f(i) in
let s-actcodei:bitstring = f21(i,fofi,q) in
let mac1ms1:mac = mac1f3(tims1,rno,tidms1,s-actcodei) in
event begMS(MS, MSi);
out(pubChannel,(MSG1,tims1,rno,tidms1,s-actcodei,mac1ms1));
in(pubChannel,(=MSG3,enctm1ms1:nonce,enctexpms1:nonce));
let dectexpms1:nonce = sdecrypttime(enctexpms1,DKms1);
event beginMSfull(MS, AS, enctm1ms1);
out(pubChannel,(MSG4,enctm1ms1));
in(pubChannel,(=MSG6,encnewrnoms1:bitstring,
encnewsimcode1:bitstring,enct1ms1:bitstring);
let dect1ms1:bitstring = sdecrypt(enct1ms1,DKms1) in
if (dect1ms1 = tims1) then
event endMSfull(AS, MS, enctm1ms1);
event endMSfull(AS, MS, enct1ms1);
(* MSi *)
let processMSi =
in(PubChannel, (xA2: host, xB2: host));
if xA2 = MSi  xB2 = MSi then
in(pubChannel,(=MSG1,tims1msi:nonce,rnomsi:bitstring,  tidms1msi:ident,  s-
actcodeimsi: bitstring, mac1ms1msi:mac));
new timsi:nonce; new simcodei:bitstring; new ki:bitstring;
new imsimsi:ident; new skmsi:skey; get keys(=MSi, skmsi) in
let DKmsi:sskey = f1(timsi,skmsi) in
insert keys(MSi,DKmsi);
let tidmsi:ident = f2(imsimsi,timsi,DKmsi) in
let actcodei:bitstring = H(simcodei) in
let mac1ms1msi2:mac = mac1f3(tims1msi,rnomsi,tidms1msi, s-actcodeimsi) in
if (mac1ms1msi = mac1ms1msi2) then
let mac2msi:mac = mac2f3(tims1msi,rnomsi,tidms1msi,
s-actcodeimsi,timsi,tidmsi,yi,zi,actcodei) in
let yi:bitstring = f13(ki,imsimsi) in
let zi:bitstring = f8(ki,DKmsi,rnomsi) in
event begMSi(MSi, AS);
out(pubChannel,(MSG2,MSG1,timsi,tidmsi,yi,zi,actcodei,
mac2msi));
in(pubChannel,(=MSG5,enctm2msi:nonce,enctexpmsi:nonce,
encnewrnomsi:bitstring,encnewsimcodeimsi:bitstring,
encDKms1asmsi:bitstring,pimsi:bitstring,mac3iasmsi:nonce));
let mac3imsi:mac = mac3f3(enctm2msi,enctexpmsi,
encnewrnomsi,encnewsimcodeimsi,encDkms1asmsi,pimsi) in
if (mac3iasmsi = mac3imsi) then
let pimsi2:bitstring = f12(DKmsi,imsimsi) in
if(pimsi2 = pimsi) then
let dectm2msi:nonce = decrypttime(enctm2msi,DKmsi);
let dectexpmsi:nonce = decrypttime(enctexpmsi,DKmsi);
let decnewrnomsi:bitstring = decrypt(encnewrnomsi,DKmsi);
let decnewsimcodeimsi:bitstring =
decrypt(encnewsimcodeimsi,DKmsi);
let decDKms1asmsi:bitstring =
decrypt(encDKms1asmsi,DKmsi);
event endMS(MS, MSi);
event endMSifull(AS, MSi, enctexpmsi);
(* AS *)
let processAS =
in(PubChannel, (xA3: host, xB3: host));
if xA3 = AS  xB3 = AS then
in(pubChannel,(=MSG2,MSG1:bitstring,timsias:nonce,
tidmsias:ident,yias:bitstring,zias:bitstring,actcodeias:bitstring,
mac2msias:mac));
mac2msias2:mac = mac2f3(MSG1,timsias,tidmsias,yias,zias,
actcodeias,mac2msias);
if (mac2msias2 = mac2msias) then
new skms1as:skey; new skmsias:skey;
new simcodeias:bitstring;
let actcodeias2:bitstring = H(simcodeias);
if (actcodeias2 = actcodeias) then
new k:bitstring;
let simcode1as:bitstring = f22(s-actcodeimsi,k) in
get keys(=MS, skms1as) in
get keys(=MSi, skmsias) in
let DKms1as:sskey = f1(tims1msi,skms1as) in
let DKmsias:sskey = f1(timsias,skmsias) in
let imsims1:ident = f2(tidms1,tims1msi,DKms1as) in
let imsimsi:ident = f2(tidmsi,timsias,DKmsias) in
insert keys(MS,DKms1as); insert keys(MSi,DKmsias);
new ki:bitstring; new m:bitstring;
let pias:bitstring = f12(DKmsias,imsimsi) in
let p:bitstring = f9(pias,m) in
let r1:bitstring = f7(rnomsi,p) in
let r2:bitstring = f11(zias,imsimsi) in
let r3:bitsting = f9(r2,m) in
let r:bitstring = f10(r3,r1) in
let q:bitstring = f9(yias,m) in
new tm1as:nonce; new texpas:nonce;
if (r = q) then
let enct1ms1:nonce = sencrypttime(tm1as,DKms1as) in
let enctexpas:nonce = sencrypttime(texpas,DKms1as) in
event beginMSfull(AS, MS, enctm1as);
out(pubChannel,(MSG3,enctm1as,enctexpas));
in(pubChannel,(=MSG4,enctm1ms1as:nonce));
if (enctm1ms1as = enctm1as) then
new tm2as:nonce;
new newrno:bitsting;
new newsimcode1:bitstring;
new newsimcodei:bitstring;
let enctm2as:nonce = sencrypttime(tm2as,DKmsias) in
let encnewrnoi:bitstring = sencrypt(newrno,DKmsias) in
let encnewsimcodei:bitstring =
```

sencrypt(newsimcodei,DKmsias) in
let encDKms1as:bitstring = sencrypt(Dkms1as,DKmsias) in
let mac3ias:mac = mac3f3(enctm2as,enctexpas,encnewrnoi,
encnewsimcodei,pias) in
event beginMSifull(AS, MSi, enctexpas);
out(pubChannel,(MSG5,enctm2as,enctexpas,encnewrno,
encnewsimcodei,encDKms1as,pias,mac3ias));
let encnewrno1:bitstring = sencrypt(newrno,DKms1as) in
let enct1ms1:nonce = sencrypttime(tims1msi,DKms1as) in
let encnewsimcode1:bitstring =
sencrypt(newsimcode1,DKms1as) in
event beginMSfull(AS, MS, enct1ms1);
out(pubChannel,(MSG6,encnewrno1,encnewsimcode1,
enct1ms1);
event endMSi(MSi, AS);
event endMSfull(MS, AS, enctm1as);
(* Key registration *)
let processK =
in(PubChannel, (h: host, k: skey));
if h <> MS && h <> MSi then insert keys(h,k).
(* Start process *)
process
new skms1:skey; new skmsi:skey;
insert keys(MS,skms1); insert keys(MSi,skmsi);
((!processMS) | (!processMSi) | (!processAS) | (!processK))

Output: −− Query attacker(s[]) ==> event(enableEnc)
Completing...ok, secrecy assumption verified: fact unreachable attacker(skms1[!1 = v_946])
Starting query attacker(s[]) ==> event(enableEnc)
RESULT attacker(s[]) ==> event(enableEnc) is true.
−− Query event(endMS(x1,x2)) ==> event(begMS(x1,x2))
Completing...ok, secrecy assumption verified: fact unreachable attacker(skms1[!1 = v_2080])
Starting query event(endMS(x1,x2)) ==> event(begMS(x1,x2))
RESULT event(endMS(x1,x2)) ==> event(begMS(x1,x2)) is true.
−− Query event(endMSi(x1_2500,x2_2501)) ==> event(begMSi(x1_2500,x2_2501))
Completing...ok, secrecy assumption verified: fact unreachable attacker(skmsi[!1 = v_3257])
Starting query event(endMSi(x1_2500,x2_2501)) ==> event(begMSi(x1_2500,x2_2501))
RESULT event(endMSi(x1_2500,x2_2501)) ==> event(begMSi(x1_2500,x2_2501)) is true.
−− Query not attacker(DK[])
Completing...ok, secrecy assumption verified: fact unreachable attacker(DKms1[!1 = v_4332])
Starting query not attacker(DK[])
RESULT not attacker(DK[]) is true.
−− Query not attacker(s[])
Completing...ok, secrecy assumption verified: fact unreachable attacker(skms1[!1 = v_5378])
Starting query not attacker(s[])
RESULT not attacker(s[]) is true.
−− Query inj-event(endMSfull(x_1138,y_1139,z)) ==> inj-event(beginMSfull(x_1138,y_1139,z))
Completing... ok, secrecy assumption verified: fact unreachable attacker(DKms1[])
ok, secrecy assumption verified: fact unreachable attacker(DKmsi[])
Starting query inj-event(endMSfull(x_1138,y_1139,z)) ==> inj-event(beginMSfull(x_1138,y_1139,z))
goal reachable: $begin(beginMSfull(AS[], MS[], enctm1as[xA3 = AS[], DKms1as = DKms1[], xB1 = MS[], !1 = endsid_1191]), enctm1ms1 = enctm1as[xA3 = AS[], DKms1as = DKms1[], xB1 = MS[], !1 = endsid_1191], DKms1 = DKms1as[], xB1 = MS[], xA3 = AS[], @sid = @sid_1192, @occ11 = @occ_cst)- > end(endsid_1191, endMSfull(AS[], MS[], enctm1as[xA3 = AS[], DKms1as = DKms1[], xB1 = MS[], !1 = endsid_1191]))$
RESULT inj-event(endMSfull(x_1138,y_1139,z)) ==> inj-event(beginMSfull(x_1138,y_1139,z)) is true.
−− Query inj-event(endMS(x_3230,y_3231)) ==> inj-event(beginMS(x_3230,y_3231))
Completing... ok, secrecy assumption verified: fact unreachable attacker(DKms1[])
ok, secrecy assumption verified: fact unreachable attacker(DKmsi[])
Starting query inj-event(endMS(x_3230,y_3231)) ==> inj-event(beginMS(x_3230,y_3231))
goal reachable: $begin(beginMS(AS[], MS[]), enctm1ms1 = enctm1as[xA3 = AS[], DKms1as = DKms1[], xB1 = MS[], !1 = endsid_4353], DKms1 = DKms1as[], xB1 = MS[], xA3 = AS[], @sid = @sid_4354, @occ8 = @occ_cst)- > end(endsid_4353, endMS(AS[], MS[]))$
RESULT inj-event(endMS(x_3230,y_3231)) ==> inj-event(beginMS(x_3230,y_3231)) is true.

ACKNOWLEDGMENTS

This work was supported by TCS, India and Australian Research Council Discovery Projects funding #DP150104871. K.-K. R. Choo is supported by the Cloud Technology Endowed Professorship.

REFERENCES

[1] P. Mondal and P. Desai, "An efficient SMS-based framework for public health surveillance," in *Proc. IEEE PCHT*, 2013, pp. 244-247.
[2] B. DeRenzi, "Improving community health worker performance through automated SMS," in *Proc. 5th ICDT*, 2012, pp. 25-34.
[3] J. Chen, L. Subramanian, and E. Brewer, "SMS-based web search for low-end mobile devices," in *Proc. MobiCom*, 2010, pp. 125-135.
[4] A. Castiglione, G. Cattaneo, A. Santis, and U. F. Petrillo, "SPEECH: secure personal end-to-end communication with handheld," in *Proc. ISSE - Securing Electronic Busines Processes*, 2006, pp. 287-297.
[5] A. Castiglione, R. Pizzolante, F. Palmieri, A. D. Santis, B. Carpentieri, and A. Castiglione, "Secure and reliable data communication in developing regions and rural areas," *Pervasive and Mobile Computing*, vol. 24, 2015, 117-128.
[6] J. Lee and Y. Oh, "A study on providing the reliable and secure SMS authentication service," in *Proc. IEEE Intl Conf on Ubiquitous Intelligence and Computing*, Bali, 2014, pp. 620-624.
[7] P. Gope and T. Hwang, "Lightweight and energy-efficient mutual authentication and key agreement scheme with user anonymity for secure communication in global mobility networks," *IEEE System Journal*, Mar. 2015, accepted.
[8] P. Traynor, W. Enck, and P. McDaniel, "Mitigating attacks on open functionality in SMS-capable cellular networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 40-53, 2009.
[9] N. Saxena and N. S. Chaudhari, "EasySMS: a protocol for end-to-end secure transmission of SMS," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 7, pp. 1157-1168, Jul. 2014.
[10] P. Vijayakumara, S. M. Ganesha, L. Deboraha, and B. Rawalb, "A new SmartSMS protocol for secure SMS communication in m-health environment," *Computers & Electrical Engineering*, pp. 1-17, 2016.
[11] Y. Yang , J. Lu, K.-K. R. Choo, and J. K. Liu, "On lightweight security enforcement in cyber-physical systems," *Lightweight Cryptography for Security and Privacy*, LNCS vol. 9542, pp. 97-112, Jan. 2016. [Online]. http://dx.doi.org/10.1007/978-3-319-29078-2_6.
[12] C. Zhang, X. Lin, R. Lu, P. Ho, and X. Shen, "An efficient message authentication scheme for vehicular communications," *IEEE Trans. on Vehicular Technology*, vol. 57, no. 6, pp. 3357-3368, 2008.
[13] J. Wu and D. R. Stinson, "Three improved algorithms for multipath key establishment in sensor networks using protocols for secure message transmission," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 929-937, 2011.
[14] J. L. Huang, L. Y. Yeh, and H. Y. Chien, "ABAKA: an anonymous batch authenticated and key agreement scheme for value-added services in vehicular ad hoc networks," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 1, pp. 248-262, 2011.
[15] S. J. Horng, S. F. Tzeng, Y. Pan, P. Fan, X. Wang, T. Li, and M. K. Khan, "b-SPECS+: batch verification for secure pseudonymous authentication in VANET," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1860-1875, Nov. 2013.
[16] G. Calandriello, P. Papadimitratos, J. P. Hubaux, and A. Lioy, "On the performance of secure vehicular communication systems," *IEEE Trans. on Dependable and Secure Computing*, vol. 8, no. 6, pp. 898-912, 2011.
[17] T. W. Chim, S. M. Yiu, L. C. K. Hui, and V. O. K. Li, "SPECS: secure and privacy enhancing communications schemes for VANETs," *Ad Hoc Networks*, vol. 9, pp. 189-203, 2011.
[18] Y. Zhou, X. Zhu, and Y. Fang, "MABS: multicast authentication based on batch signature," *IEEE Transaction on Mobile Computing*, vol. 9, no. 7, pp. 982-993, Jul. 2010.
[19] J. A. Akinyele, M. Green, and S. Hohenberger, "Machine-generated algorithms, proofs and software for batch verification of digital signature schemes," in *Proc. Cryptology*, 2013, pp. 1-41.
[20] N. Saxena and N. S. Chaudhari, "SecureSMS: a secure SMS protocol for VAS and other applications," *Journal of Systems and Software*, vol. 90, pp. 138-150, 2014.
[21] J. L. Lo, J. Bishop, and J. H. P. Eloff, "SMSSec: an end-to-end protocol for secure SMS," *Computers & Security*, vol. 27, no. 5-6, pp. 154-167, 2008.
[22] H. Rongyu, Z. Guolei, C. Chaowen, X. Hui, Q. Xi, and Q. Zheng, "A PK-SIM card based end-to-end security framework for SMS," *Computer Standards and Interfaces*, vol. 31, no. 4, pp. 629-641, 2009.

[23] C. C. Yang, Y. L. Tang, and R. C. Wang, "A secure and efficient authentication protocol for anonymous channel in wireless communications," *Applied Math. & Computation*, vol. 169, no. 2, pp. 1431-1439, 2005.

[24] A. A. Shaikh and N. S. Vani, "An extended approach for securing the short messaging services of GSM using multi-threading elliptical curve cryptography," in *Proc. ICCICT*, 2015, pp. 44-48.

[25] S. Islam, I. U. Haq, and A. Saeed, "Secure end-to-end SMS communication over GSM networks," in *Proc. IBCAST*, 2015, pp. 286-292.

[26] S. Karale, K. Pendke, and P. Dahiwale, "Survey of techniques & algorithms for SMS security," in *Proc. ICIIECS*, 2015, pp. 1-6.

[27] S. Bojjagani and V. N. Sastry, "SSMBP: a secure SMS-based mobile banking protocol with formal verification," in *Proc. WiMob*, 2015, pp. 252-259.

[28] C. C. Chang and J. S. Lee, "Efficient authentication protocols of GSM," *Computer Comm.*, vol. 28, no. 8, pp. 921-928, 2008.

[29] N. Saxena and N. S. Chaudhari, "SAKA: a secure authentication and key agreement protocol for GSM networks," *CSI Transactions on ICT*, vol. 1, no. 4, pp. 331-341, 2013.

[30] N. Saxena and N. S. Chaudhari, "NS-AKA: an improved and efficient AKA protocol for 3G (UMTS) networks," in *Proc. CSEE*, Mar. 2014, pp. 220-224.

[31] C. C. Lee, C. L. Chen, and H. H. Ou, "Extension of an efficient 3GPP authentication and key agreement protocol," *Wireless Personal Communications*, vol. 68, no. 3, pp. 861-872, 2013.

[32] Y. L. Huang, C. Y. Shen, and S. W. Shieh, "S-AKA: a provable and secure authentication key agreement protocol for UMTS networks," *IEEE Trans. on Vehicular Tech.*, vol. 60, no. 9, pp. 4509-4519, 2011.

[33] N. Saxena, J. Thomas, and N. S. Chaudhari, "ES-AKA: an efficient and secure authentication and key agreement protocol for UMTS networks," *Wireless Personal Communications*, vol. 84, no. 3, pp. 1981-2012, Oct. 2015.

[34] N. Saxena and N. S. Chaudhari, "Secure-AKA: an efficient AKA protocol for UMTS networks," *Wireless Personal Communications*, vol. 78, no. 2, pp. 1345-1373, Sep. 2014.

[35] A. Castiglione, G. Cattaneo, G. D. Maio, and F. Petagna, "SECR3T: secure end-to-end communication over 3G telecommunication networks," in *Proc. International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Seoul, 2011, pp. 520-526.

[36] F. Hadiji, F. Zarai, and A. Kamoun, "Authentication protocol in 4th generation wireless networks," in *Proc. WOCN*, 2009, pp. 36-39.

[37] G. M. Kϕien, "Mutual entity authentication for LTE," in *Proc.* 7^{th} *IWCMC*, 2011, pp. 689-694.

[38] Y. W. Chen, T. Wang, K. H. Chi, and C. C. Tseng, "Group-based authentication and key agreement," *Wireless Personal Communications*, vol. 62, no. 4, pp. 965-979, 2012.

[39] C. Lai, H. Li, R. Lu, and X. Shen, "SE-AKA: a secure and efficient group authentication and key agreement protocol for LTE networks," *Computer Networks*, vol. 57, no. 17, pp. 3492-3510, 2013.

[40] M. S. A. Khan and C. J. Mitchell, "Improving air interface user privacy in mobile telephony," *Cryptography and Security*, Cornell University Library, Apr. 2015. [Online]. http://arxiv.org/abs/1504.03287

[41] M. Toorani and A. Beheshti, "SSMS - A secure SMS messaging protocol for the m-payment systems," in *Proc. IEEE Symposium on Computers and Communications*, 2008, pp. 700-705.

[42] A. D. Santis, A. Castiglione, G. Cattaneo, M. Cembalo, F. Petagna, and U. F. Petrillo, "An extensible framework for efficient secure SMS," in *Proc. Int. Conf. CISIS*, 2010, pp. 843-850.

[43] A. Castiglione, G. Cattaneo, M. Cembalo, A. Santis, P. Faruolo, F. Petagna, and U. F. Petrillo, "Engineering a secure mobile messaging framework," *Computers & Security*, vol. 31, no. 6, pp. 771-781, 2012.

[44] M. Thomas and V. Panchami, "An encryption protocol for end-to-end secure transmission of SMS," in *Proc. International Conference on Circuits, Power and Computing Technologies*, 2015, pp. 1-6.

[45] N. Saxena and N. S. Chaudhari, "SecureSMS: A secure SMS protocol for VAS and other applications," *The Journal of Systems and Software*, vol. 90, pp. 138-150, 2014.

[46] J. L.-C. Lo, J. Bishop, and J. Eloff, "SMSSec: an end-to-end protocol for secure SMS," *Computers & Security*, vol. 27, pp. 154-167, 2008.

[47] H. Rongyu, Z. Guolei, C. Chaowen, X. Hui, Q. Xi, and Q. Zheng, "A PK-SIM card based end-to-end security framework for SMS," *Elsevier Computer Standards & Interfaces*, vol. 31, pp. 629-641, 2013.

[48] N. Saxena and N. S. Chaudhari, "VAS-AKA: an efficient batch verification protocol for value added services," in *Proc. IEEE SMC*, 2013, pp. 1560-1565.

[49] S. H. Al-Bakri, M. L. Mat-Kiah, A. A. Zaidan, B. B. Zaidan, and G. M. Alam, "Securing peer-to-peer mobile communications using public key cryptography: New security strategy," *International Journal of the Physical Sciences*, vol. 6, no. 4, pp. 930-938, 2011.

[50] M. L. Mat-Kiah, S. H. Al-Bakri, A. A. Zaidan, B. B. Zaidan, and M. Hussain, "Design and develop a video conferencing framework for real-time telemedicine applications using secure group-based communication architecture," *J. of Medical Systems*, pp. 38-133, 2014.

[51] A. Medani, A. Gani, O. Zakaria, A. A. Zaidan, and B. B. Zaidan, "Review of mobile short message service security issues and techniques towards the solution," *Scientific Research and Essays*, vol. 6, no. 6, pp. 1147-1165, Apr. 2011.

[52] A. Castiglione, F. Palmieri, U. Fiorec, A. Castiglione, and A. D. Santis, "Modeling energy-efficient secure communications in multi-mode wireless mobile devices," *Journal of Computer and System Sciences*, vol. 81, no. 8, 2015, pp. 1464-1478.

[53] A. Castiglione, A. D. Santis, F. Palmieri, and U. Fiore, "An energy-aware framework for reliable and secure end-to-end ubiquitous data communications," in *Proc. International Conf. on Intelligent Networking and Collaborative Systems*, 2013, pp. 157-165.

[54] N. Saxena and N. S. Chaudhari, "A secure approach for SMS in GSM network," in *Proc. CUBE*, 2012, pp. 59-64.

[55] N. Saxena and N. S. Chaudhari, "A secure digital signature approach for SMS security," *International Journal of Computer Applications: SI on IP Multimedia Communications*, pp. 98-102, 2011.

[56] A. Merlo, M. Migliardi, N. Gobbo, F. Palmieri, and A. Castiglione, "A denial of service attack to UMTS networks using SIM-less devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 3, pp. 280-291, 2014.

[57] N. Saxena and N. S. Chaudhari, "Secure-AKA: An efficient AKA protocol for UMTS networks," *Wireless Personal Communications*, vol. 78, no. 2, pp. 1345-1373, Sept. 2014.

[58] E. Oliver, Design and Implementation of a Short Message Service Data Channel for Mobile Systems, Technical Report CS-2007-42, 2007, 10 pages. [Online]. https://cs.uwaterloo.ca/research/tr/2007/CS-2007-42.pdf.

[59] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, pp. 612-613, 1979.

[60] WLAN Pseudonym Generation for EAP-SIM/AKA, 3GPP TSG SA WG3 Security, S3-020654, Nov. 2002. [Online]. ftp://www.3gpp.org/tsg_sa/WG3_Security/TSGS3_26_Oxford/Docs/PDF/S3-020654.pdf.

[61] A. Eshak, H. M. Harb, and M. Ezz, "Scalable intrusion detection system for cellular networks," in *Proc. IEEE ASID*, 2013, pp. 1-8.

[62] A. Berkopec, "HyperQuick algorithm for discrete hypergeometric distribution," *J. of Discrete Algorithms*, vol. 5, no. 2, pp. 341-744, 2007.

[63] K. Kurosawa, S. Obana, and W. Ogata, "t-cheater identifiable (k, n) threshold secret sharing schemes," in *Proc. Advances in Cryptology*, LNCS 963, 1995, pp. 410-423.

[64] E. Kashefi, D. Markham, M. Mhalla, and S. Perdrix, "Information flow in secret sharing protocols," *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, vol. 9, pp. 87-97, 2009.

[65] L. Harn and C. Lin, "Detection and identification of cheaters in (t, n) secret sharing scheme," *Designs, Codes and Cryptography*, vol. 52, no. 1, pp. 15-24, 2009.

[66] N. Golde, K. Redon, and J.-P. Seifert, "Let me answer that for you: exploiting broadcast information in cellular networks," in *Proc. 22nd USENIX Security Symposium*, Aug. 2013, pp. 33-48.

[67] How fast is 4G? - 4G speeds and UK network performance. [Online]. http://www.4g.co.uk/how-fast-is-4g.

[68] D.-M. Ciuraru, L. Hilohi, A. Mercier, and X. Lagrange, "SMS over LTE: services, architecture and protocols," 13218, 2013, pp. 62. [Online]. https://hal.archives-ouvertes.fr/hal-00814264/document.