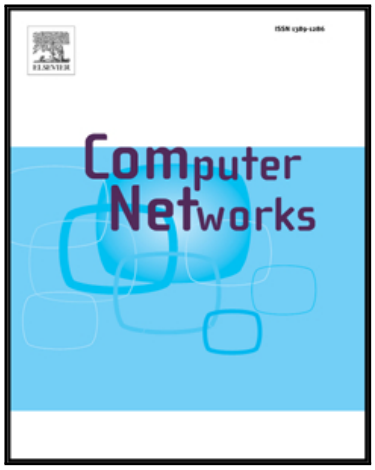
Journal Pre-proof



MSYM: A Multichannel Communication System for Android Devices

Wenjie Wang, Donghai Tian, Weizhi Meng, Xiaoqi Jia, Runze Zhao, Rui Ma

PII: S1389-1286(19)30998-3
DOI: https://doi.org/10.1016/j.comnet.2019.107024
Reference: COMPNW 107024

To appear in: *Computer Networks*

Received date: 6 August 2019
Revised date: 29 October 2019
Accepted date: 21 November 2019

Please cite this article as: Wenjie Wang, Donghai Tian, Weizhi Meng, Xiaoqi Jia, Runze Zhao, Rui Ma, MSYM: A Multichannel Communication System for Android Devices, *Computer Networks* (2019), doi: https://doi.org/10.1016/j.comnet.2019.107024

Highlights

- We propose a multichannel communication mechanism for mobile devices to secure sensitive data transfer.
- We exploit the Android VpnService interface to intercept network data and then split it into different fragments. To improve the data communication security, the fragments are transferred via multiple transmission channels.
- We design and implement a prototype of system based on the Android system to transfer the sensitive data securely.

MSYM: A Multichannel Communication System for Android Devices

Wenjie Wang[a], Donghai Tian[a,c,*], Weizhi Meng[b], Xiaoqi Jia[c,d], Runze Zhao[a], Rui Ma[a]

[a]*Beijing Key Laboratory of Software Security Engineering Technique, Beijing Institute of Technology, Beijing 100081, China*
[b]*Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark*
[c]*Key Laboratory of Network Assessment Technology, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100049, China*
[d]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China*

Abstract

Conventional mobile communication systems often use one single channel for data transmission, i.e., mobile devices use cellular network to transfer multimedia information. However, if attackers successfully hijack the single transmission channel, they can recover the communicated data. Focused on this issue, we introduce a Multichannel Communication System (MSYM), which aims to improve the data communication security for Android devices. The key idea of our approach is to leverage the diversity of communication mechanisms (e.g., Wi-Fi/cellular network, Bluetooth, and SMS) for transferring sensitive data in a secure way. More specifically, we use the VpnService interface provided by the Android platform to intercept the network data delivered by a sender program. Then, we split the network data into different fragments and improve the security by disordering and encrypting them via multiple transmission channels. When the target Android device receives the data fragments from different channels, it can decrypt and reorder them to reassemble the original data. In the end, we reuse the VpnService interface to inject the network data into the receiver program. Our approach can be deployed in Android devices to secure communication without the need of

*Corresponding author
Email address: donghaitad@gmail.com (Donghai Tian)

modifying the communication programs. In the evaluation, as a proof of concept, we implemented our approach on Android system. The experimental results show that our prototype system can secure data transmission with moderate performance cost.

Keywords: Mobile communication, Multiple transmission, Network data, Android device, Eavesdropping attacks

1. Introduction

Mobile communication plays an increasingly important role in recent years, as more and more people exchange information via mobile devices. For example, over 2 billion people are using the instant messaging apps (e.g., WeChat and WhatsApp) on their mobile phones for exchanging messages, pictures and videos [6]. In addition, 90% of enterprises utilize mobile communication to boost productivity and streamline various business processes. Traditionally, most mobile devices only use one single wireless transmission channel (e.g., cellular network) for mobile communication. However, due to the openness of the wireless channel, it is possible for advanced attackers to eavesdrop the transmission and recover the sensitive data. For instance, adversaries can make use of bogus base stations to conduct man-in-the-middle (MITM) attack on modern Samsung mobile devices and obtain the sensitive information [23].

To prevent the communication data from being eavesdropped and recovered, a conventional solution is to leverage cryptographic protocols to perform data encryption and endpoint authentication. For example, the Wi-Fi Protected Access (WPA) and its subsequent standards (WPA2, WPA3) attempt to secure the wireless networks by applying multiple security protocols [13]. However, they may still suffer from eavesdropping attacks like the KRACK [35] attack. This attack shows that an attacker can exploit the vulnerability of WPA2 to read previously encrypted information. On the other hand, most mobile applications adopt the client-server model, which requires that all the communication data between two or more clients should be firstly forwarded to a central server. Nevertheless, some service providers may not be fully trusted. Taking the messaging server as an example, once it is hacked by adversaries, the recorded information would be leaked.

To address the above problems, in this paper, we design and implement MSYM - a Multiplex channel (use Multichannel for short) Communication

System by leveraging the diversity of communication mechanisms on mobile devices. The basic idea of our approach is to intercept the network data and then transfer it via multiple data transmission channels. In comparison with other solutions, MSYM can protect the sensitive data from unauthorized access without the need of implementing complicated cryptographic protocols. The workflow of MSYM can be divided into several steps. First, we use the VpnService interface [9] provided by Android platform to intercept the network data delivered by a sender program. Then our system splits it into different fragments, which have to be disordered and encrypted before transmission. Later, these fragments are transferred via multiple transmission channels, including Wi-Fi/cellular network, Bluetooth, and Short Message Service (SMS). When the target Android device receives the data fragments from different transmission channels, it has to decrypt and reorder them to recover the original network data. In the end, we reuse the VpnService interface to inject the network data into the receiver program.

With the multichannel communication mechanism, each transmission channel only retains partial network data. That means attackers can only obtain partial data / fragments if they eavesdrop over one channel. In this case, they cannot recover the whole data based on these partial fragments. Further, even if advanced attackers can capture more data fragments from multiple channels, it is still difficult for them to retrieve the original data as fragments are disordered and encrypted. Similarly, the service providers could only obtain partial communication data; thus, they have no idea on how to recover the original data. Hence our approach can be used to secure sensitive data transmitted on Android devices against network eavesdropping attacks.

To exploit the performance of our approach - MSYM, we implemented a prototype based on Android platform. To improve the reliability of multichannel communication, we design a channel switch and recovery mechanism, which can automatically switch the transmission channel when the network speed of one channel becomes too low (or it has a problem). In the evaluation, we leverage an open-source Android instant messaging app and transfer its network data over MSYM. Our experimental results show that the communication data between two Android devices can be transferred via multiple channels effectively, and that our system can perform the channel switch automatically when its quality of service (QoS) becomes poor. When the previous channel becomes normal, our system can switch back and recover such channel. Due to the benefits provided by multichannel communication, MSYM may incur a moderate performance cost on Android devices. Overall,

our contributions can be summarized as follows:

- We propose a multichannel communication mechanism - MSYM for mobile devices, which can be used for securing sensitive data transferred on Android devices.
- We exploit the Android VpnService interface to intercept network data and then split it into different fragments, which are disordered and encrypted. Then we utilize the multiple data transmission channels to send the fragments in a secure way.
- We design and implement a prototype system based on Android system. The evaluation results demonstrate that our prototype can transfer the data via multiple transmission channels effectively with moderate cost.

The rest of this paper is organized as follows. Section 2 analyzes the background and motivation of our paper. Section 3 introduces the related work. Section 4 and Section 5 present the design and implementation of MSYM, respectively. Section 6 evaluates the performance of MSYM with the communication applications. Section 7 discusses MSYM's limitations and possible solutions. Section 8 concludes this paper.

2. Background and Motivation

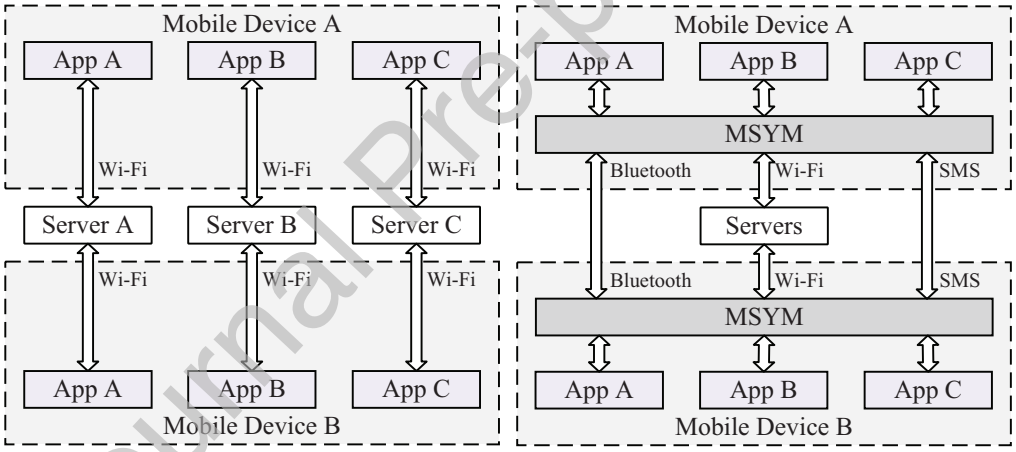
In this section, we begin with introducing the traditional mobile communication mechanisms as well as potential security issues. Then, we propose a multichannel mechanism and analyze the enhanced security properties for mobile communication.

2.1. Traditional Mobile Communication

Figure 1(a) illustrates the traditional mobile communication mechanism, where mobile devices only use one single transmission channel for wireless communication. For instance, smartphones send multi-media information via the Wi-Fi or cellular network, and most mobile applications adopt the client-server model for data communication. That is, before the data transmission between two mobile devices, mobile applications have to connect to the central server at first. Then, the server receives the network data from one application, and forwards it to another application on the target mobile device.

As the wireless transmission channel is open, advanced attackers can eavesdrop the communication data. Typically, attackers can use bogus base station or malicious wireless router to harvest network data. If the data is not encrypted, attackers can easily obtain the sensitive information. Even if the network data is encrypted, some advanced attackers can still recover the original information. This is because some cryptographic protocols like [22, 1, 2, 35] may have implementation vulnerabilities that could be exploited by attackers to recover the encrypted data.

In addition, as most mobile applications depend on the central server for data transmission, the server may become a target for cyber-criminals. In practice, some servers record the transmitted and historical data. Once the server is compromised by an attacker, the private information would be leaked. Moreover, some untrusted providers may track, identify and profile users by analyzing the transmitted data.



(a) Traditional communication mechanism (b) Multichannel communication mechanism

Figure 1: Comparison of traditional and multichannel communication mechanisms

2.2. Multichannel Mobile Communication

To address the limitations of traditional mobile communication, we propose a multichannel mobile communication solution. This solution leverages the diversity of communication mechanisms (including Wi-Fi/cellular network, Bluetooth, and SMS) to secure mobile communication on mobile devices. Figure 1(b) shows how the multichannel mobile communication works.

First, it needs to intercept the communication data sent by a mobile program. Then, the intercepted data is split into different fragments, which are distributed into different channels randomly. Then the fragments are transferred via multiple transmission channels. When the target mobile device receives the data fragments from different channels, it needs to reassemble the fragments to recover the original communication data.

Compared with the traditional mobile communication, our multichannel solution raises the difficulty for advanced attackers to acquire the sensitive communication data. With the multichannel transmission scheme, eavesdropping attackers on one transmission channel can only obtain the partial communication data. As a result, attackers have to eavesdrop all the transmission channels in order to recover the original data.

Even if the attackers can fully control the central server, it is still difficult for them to recover the original communication data due to the split fragments. As the fragments are disordered, the attackers have no idea on how to reassemble and recover the original data. Further, as some data fragments are not transferred via the Wi-Fi (or cellular network), the attackers cannot obtain the whole communication data by only eavesdropping the server.

3. Related Work

In the literature, many solutions have been proposed to enhance the mobile communication security. Mobile Virtual private network (VPN) is an effective way to establish secure and reliable connection between two mobile devices [3, 12]. In general, the mobile VPN can encrypt the network packets based on IPsec [34], Secure Sockets Layer (SSL) [10], or Transport Layer Security (TLS) [29] protocols. Additionally, a SIP-based mobile VPN solution is proposed for real-time applications, which comprises of several protocols (e.g., SIP and cRTP) to provide secure VPN services [18, 31]. In contrast to these solutions, our MSYM can effectively protect the communication security without using a complex cryptographic protocol. It can also prevent the service providers from accessing the sensitive communication data.

Covert channel [15] is another effective way to transfer confidential information in untrusted mobile networks. It hides covert messages by exploiting an authorized overt communication as the carrier medium for stealth communication [41]. Many schemes [7, 5, 11, 17, 40] based on covert channel have been developed to secure the mobile communication. For example, Zhang *et al.* [42] propose a method that re-arranged packets over mobile networks

to build covert channels. Moreover, Zhang *et al.* also present a packet-reordering strategy for Voice over Long-Term Evolution (VoLTE) [43], which could improve the undetectability and robustness of communication by reordering voice and video packets. Tan *et al.* [4] present an end-to-end covert channel solution for mobile networks. This solution builds a covert channel based on VoLTE video stream via dropping out some specific packets. Different from these covert channel solutions, our MSYM protects the communication security by exploiting the multiple transmission channels without the need of modifying the Android system.

Recently, the Android VpnService interface has been used for different research purposes. Some researchers leverage it to detect the network delay [38] and the traffic differentiation [19] in mobile networks. For example, MopEye [38] employs the VpnService to measure the network round-trip delay of each app of Android system. Baidu's TrafficGuard [16] optimizes the network traffic by using the VPN to connect a remote server. Some researchers (e.g., [30, 27, 28]) focus on using the VpnService to detect any privacy leakage. PrivacyGuard [32] utilizes the VpnService interface to intercept the network traffic and then analyzes whether there is sensitive data. Haystack [27] leverages the VpnService mechanism to measure real-world mobile network traffic. AntMonitor [30] takes advantage of the VpnService interface for efficient passive on-device mobile network monitoring. UpDroid [33] exploits the VpnService to detect the sensitive networking behavior of Android applications. Different from these VPN applications, to the best of our knowledge, our MSYM is the first system that leverages the VpnService mechanism for multichannel communication on Android devices.

Some methods use the client-server VPN model, which transmits network data to a remote server for data processing or logging. Meddle [26] is a representative work to adopt this model for interposing all Internet traffic on a mobile device. Recon [28] extends this work to identify privacy leakage via mobile network. Since both of them adopt the client-server VPN model, all network packets need to be sent to the remote server. This model may result in the additional network delay and data leakage during transmission. MSYM adopts the local VPN model, which processes network data on the local device by maintaining a local TCP/IP stack.

4. Design

4.1. Overview

Based on the multichannel communication mechanism, the general architecture of our system is shown in Figure 2. There are three modules in MSYM: data interception, data processing, and data transmission. On the message sender, the data interception module utilizes the VpnService interface to intercept the outgoing data sent by IM applications. After getting the outgoing data, the data processing module splits the data into several fragments. To improve the data security, these data fragments have to be disordered and encrypted. Then, the data transmission module transfers these fragments via three data transmission channels, including Wi-Fi/cellular network, Bluetooth, and SMS. On the message receiver, the data transmission module receives the data fragments from all the channels. The data processing module then has to decrypt & reorder these fragments and reassemble the original data. Finally, the target application can receive the data, after the data interception module injects the reassembled data to the VpnService interface.

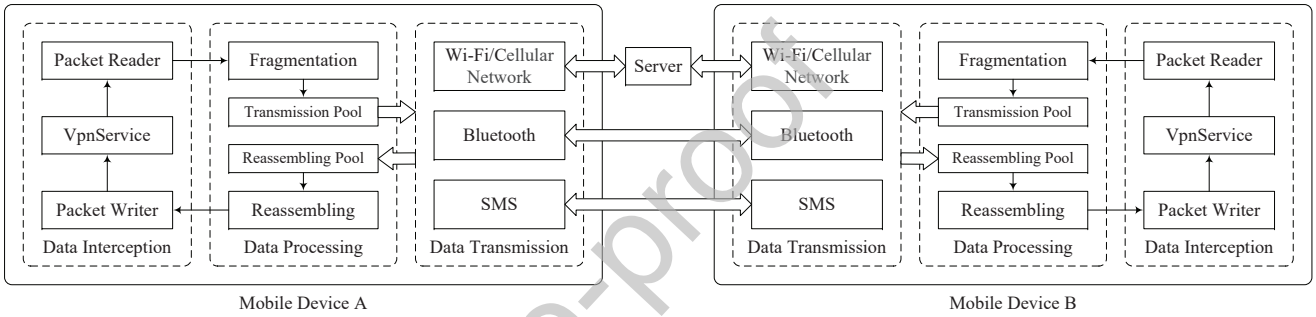


Figure 2: An overview of MSYM's architecture

The basic workflow of our multichannel communication mechanism can be summarized as follows:

1) The data interception module traps IP packets sent by a sender program on the mobile device. Then, it reads and parses these packets to extract their payloads.

2) The data processing module splits each payload into different data fragments. To prevent these fragments from being reassembled by an attacker, this module disorders and encrypts them. Afterwards, it puts these fragments into the data transmission pool.

3) The data transmission module starts transferring data by distributing the fragments from the data transmission pool to the transmission channels in a random manner. There are three transmission channels at this module: Wi-Fi/cellular network, Bluetooth and SMS.

4) When the target device receives the communication data through the data transmission channels, the data can be stored in the data reassembling pool.

5) The data processing module can decrypt and reorder the data from the data reassembling pool to recover the original data.

6) The data interception module can reconstruct an IP packet based on the reassembled data and then inject it into the receiver program.

4.2. Data Interception

Before transferring the data via multiple transmission channels, we have to firstly intercept the network data sent by the target communication program running on the Android device. For this purpose, one potential solution is to modify the target program. However, it may not be compatible with the code signing mechanism, and may make more engineering efforts for modifying the code. Instead, in this work, we use the VpnService interface to intercept the network data, without the need of root privilege and application rewriting.

4.2.1. VpnService

Android VpnService interfaces exploit the TUN virtual network device to intercept the network data. When the VpnService is enabled, Android system can create a virtual network interface (VNI). After configuring the addresses and routing rules, all outgoing network packets can be forwarded to this VNI device. To facilitate accessing this device, Android system provides a file descriptor for the target application. There are two major operations to handle the descriptor: read and write. The former operation retrieves outgoing packets routed to the VNI device, while the latter operation writes incoming packets to the descriptor such that these packets can arrive at target applications through the VNI. Accordingly, there are two components in the data interception module: Packet Reader and Packet Writer.

4.2.2. Packet Reader

The packet reader reads and parses outgoing network packets from the VNI device. As the VNI runs on the Internet Protocol (IP) layer, the packets should always start with IP headers. Intuitively, an IP datagram can be used for the TCP/UDP connection, in this work, we mainly focus on the TCP connection, because it is more popular and complicated in practice.

Figure 3 shows the process of reading and parsing an IP packet. To improve the network performance, the packet reader can create two threads: reader thread and parser thread. The reader thread gets the IP packets from VNI. Instead of analyzing the packets instantly, it puts the packets into the FIFO ring buffer. Then the parser thread analyzes the packets and dispatches them for future processing. Thanks to the wait-free property of the ring buffer, the reader and parser thread can perform their tasks concurrently.

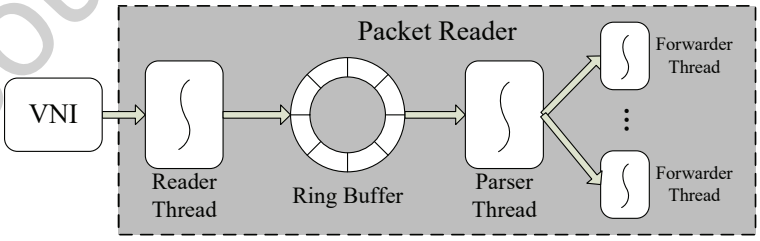


Figure 3: The process of packet reader

In general, a network packet can be divided into three parts: IP header, TCP header, and TCP payload. After extracting the TCP payload from

a packet, the parser thread has to pass it to the data processing module. To this end, the parser thread dynamically creates a forwarder thread for each TCP connection. In this case, a packet from the same TCP connection can be dispatched to its corresponding forwarder thread, aiming to maintain TCP states and data processing.

4.2.3. Packet Writer

The packet writer is responsible for reconstructing IP datagrams and injecting them into the VNI, so that the communication application can receive the network data. As shown in Figure 4, the packet writer includes multiple forwarder threads, multiple reconstruction threads, and a writer thread. The forwarder threads are used for receiving data from the data transmission module. As the data transmission module uses regular TCP socket for data transmission, the forwarder threads can receive the TCP payloads only. However, the writer thread has to inject the raw IP datagrams (not payload) into the VNI. In the end, the reconstruction threads are created. For each TCP connection, the reconstruction thread has to construct the corresponding IP datagrams based on the TCP connection states maintained in the forwarder thread.

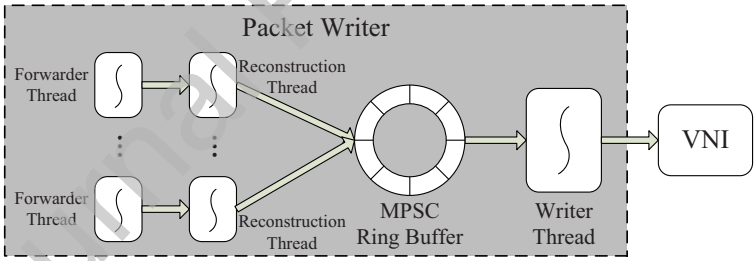


Figure 4: The data flow of packet writer

To generate a valid IP datagram, the reconstruction thread has to reconstruct the valid IP and TCP headers for each TCP payload. After generating the IP datagrams, the writer thread can perform the network data injection for the target receiver program. To ensure the reconstruction threads and writer thread working in parallel, we use the Lock-free multi-producer single-consumer (MPSC) ring buffer, which utilizes the read-modify-write atomic operations for data synchronization.

4.3. Data Processing

The main task of the data processing module is to split the outgoing TCP payload and resemble the incoming data. Figure 5 shows the basic workflow of this task. For the outgoing IP packets, the forwarder thread splits their TCP payloads into various data fragments and then forwards them into the data transmission pool with random order. To facilitate the data reassembling, a data header is added for each data fragment. The data transmission module can forward the incoming data into the data reassembling pool. Then, the reconstruction thread will reassemble the data into raw TCP payloads.

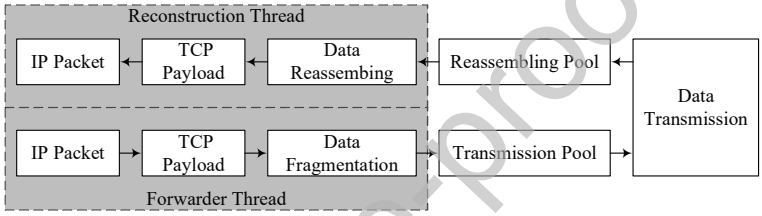


Figure 5: The work flow of data processing

4.3.1. Data Fragmentation

The data fragmentation procedure aims to generate encrypted and irregular data fragments. To this end, there are three phases in this procedure: data splitting, header encryption, and data disorder. The forwarder thread first splits the TCP payload into several fragments. To reassemble these fragments on the target device, each fragment is marked with a two-byte header (as shown in Figure 6(a)). This header identifies the length and ID of a fragment, so that the data reassembling procedure knows the original location of this fragment. To enhance the security of data transmission, the forwarder thread encrypts each fragment header, and gathers these fragments into the data transmission pool in a random order. In this case, even if attackers can obtain all the fragments, they do not know the original position of each fragment. This can greatly increase the cracking difficulty for even advanced attackers.

Algorithm 1 shows the procedure of data fragmentation. Before splitting a TCP payload, we should first know how many data fragments that the payload needs to be split (Line 2). For each intended fragment, we identify the fragment length (Line 6) and then encrypt the length flag (Line 7). We

also input this encrypted length to the associated $fragments$ array entry. Each $fragments$ array entry stores a data fragment. Similarly, we encrypt the fragment ID that records the original location of this fragment. Then, we input it to the $fragments$ array entry as well (Line 8). After that, we input the corresponding part of the payload to the $fragments$ array entry (Line 9). When all the fragments are fully prepared, we generate a random sequence (Line 12) by using the Fisher-Yates shuffle algorithm [36]. According to the generation sequence, we input all the data fragments to the *result* array (Line 14-15).

Algorithm 1 Data Fragmentation

Input: A TCP payload: *payload*.
Output: An array of unordered data fragments: *result*.

1: **function** FRAGMENTATION(*payload*)
2: $n \leftarrow getFragmentNumber(payload)$;
3: $fragments[] \leftarrow$ an empty two-dimensional array; ▷ The fragments[] array is used to store the data fragments
4: $start \leftarrow 0$; ▷ Start is the index of the fragment
5: **for** each $id \in [1, n]$ **do**
6: $len \leftarrow getFragmentLength(start, id, n, payload)$;
7: $fragments[id][0] \leftarrow encrypt(len)$; ▷ Encrypt the fragment length and put it into the $fragments[]$
8: $fragments[id][1] \leftarrow encrypt(id)$; ▷ Encrypt the original position id and put it into the $fragments[]$
9: $fragments[id][2 : 2 + len] \leftarrow payload[start : start + len]$; ▷ Put the corresponding part of the payload into the $fragments[]$
10: $start \leftarrow start + len$;
11: **end for**
12: $ran[] \leftarrow$ randomSequence$(1, n)$; ▷ Generate a random sequence in the range of 1 and n
13: $result[] \leftarrow$ an empty array;
14: **for** each $i \in [1, n]$ **do**
15: $result[i] \leftarrow fragments[ran[i]]$; ▷ Put the data fragment numbered $ran[i]$ into the $result[]$
16: **end for**
17: **return** *result*;
18: **end function**

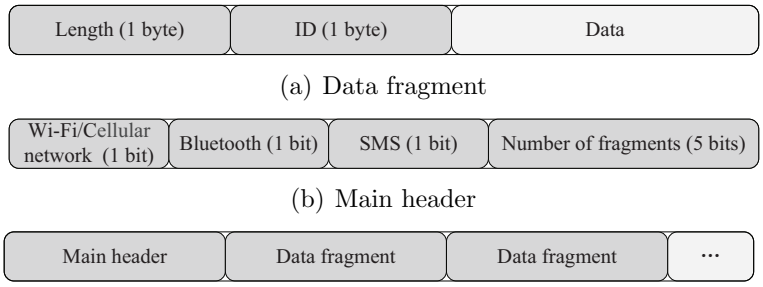


Figure 6: The data format for data processing and transmission

A Motivating Example. Figure 7 shows a simple example to illustrate the workflow of data fragmentation. Firstly, the raw data from the packet reader is split into 16 fragments numbered sequentially from 1 to 16. Each fragment adds an encrypted header to identify its length and original position. Secondly, a random sequence is generated between 1 and 16, such as {5, 13, 3, 8, ..., 15}. According to the order of the generated sequence, we input these fragments to the data transmission pool. Thus, these data fragments stored in the data transmission pool could be numbered as {5, 13, 3, 8, ..., 15}. In the end, these fragments have to be distributed to the remote mobile device via aforementioned three data transmission channels.

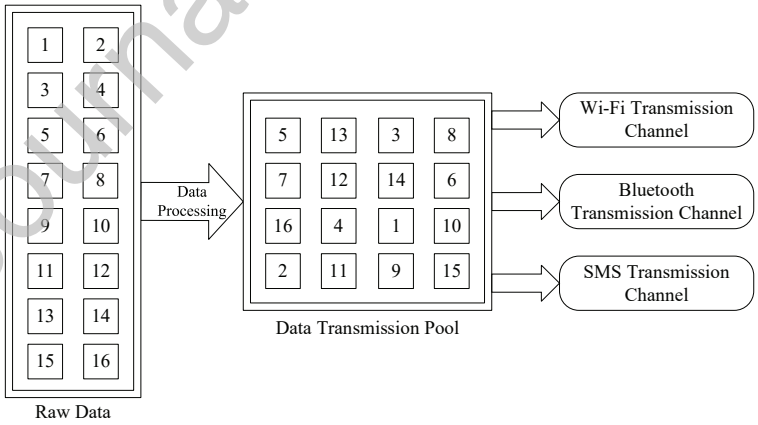


Figure 7: An example of data fragmentation

4.3.2. Data Pool

After receiving a data fragment, a straightforward method is to deliver it to the data transmission module immediately. However, this may introduce considerable performance cost for real-time transmission when there are a large number of fragments. To address this problem, we apply a data transmission pool to cache outgoing data fragments for later transmission. Similarly, we use a data reassembling pool to store the incoming data for later reassembling. Thanks to the data structures, the forwarder thread and reconstruction thread can process subsequent data without waiting for the current data being sent or reassembled.

In the data transmission module (§4.4), there are three sender threads retrieving data from the data transmission pool and three receiver threads inputting data to the data reassembling pool. To avoid the thread collision problems caused by several threads accessing a data pool simultaneously, we take advantage of the thread-safe blocking queue, which supports multi-thread access based on internal locks. In particular, it blocks threads retrieving an element when it is empty, and blocks threads storing an element when the space is full.

4.3.3. Data Reassembling

Due to the data fragmentation, the incoming data received from the multiple transmission channels is encrypted and disordered. By considering the transmission performance, we combine the data fragments in the data transmission module (§4.4). For data reassembling, each incoming data contains a main header and several data fragments, as shown in Figure 6(c). To ensure the communication application can get the raw data, the incoming data should be reassembled into the raw TCP payload.

Algorithm 2 shows the procedure of data reassembling. Compared with the data fragmentation procedure, there are two opposite operations in the data reassembling procedure: header decryption, and data reordering. After the reconstruction thread retrieves the data from the data reassembling pool, it has to first decrypt the first byte of the incoming data aiming to get the main header (Line 2). Based on this main header, we can know the number of data fragments (Line 3). For each fragment, we decrypt the first two bytes of the fragment to recover its length (Line 7) and the original position id (Line 8). Then, we extract the raw data of this fragment and store it into the associated $fragments$ array entry (Line 9). After recovering all the data fragments, we reorder them in a sequence to recover the original payload

(Line 12). Finally, the reconstruction thread utilizes the reassembled payload to construct the IP datagram.

Algorithm 2 Data Reassembling

Input: The incoming data: *data*.
Output: The reassembled data: *result*.

```
1: function REASSEMBLINGDATA(data)
2:     mainHeader ← decrypt(data[0]);      ▷ Decrypt the first byte of the incoming data to get the main header
3:     n ← getFragmentNumber(mainHeader);
4:     fragments[] ← an empty array;   ▷ The fragments[] array is used to store the data fragments
5:     index ← 1;
6:     for each i ∈ [1, n] do
7:         len ← decrypt(data[index++]);  ▷ Get the data fragment length
8:         id ← decrypt(data[index++]);  ▷ Get the original position id of the data fragment
9:         fragments[id] ← data[index : index + len];   ▷ Put the raw data fo the fragment into its original position of the fragments[]
10:        index ← index + len;  ▷ Get the index of the next data fragment
11:    end for
12:    result[] ← fragments[1] : fragments[2] : ... : fragments[n];   ▷ Combine the data fragments into the result[] in sequence
13:    return result;
14: end function
```

4.4. Data Transmission

In the data transmission module, there are three different channels, including Wi-Fi/cellular network, Bluetooth, and SMS. For each channel, the corresponding communication instance should be launched to communicate with the remote device. There are two threads for each communication instance: sender thread and receiver thread. The former is responsible for sending the outgoing data to the remote mobile device. The main task of the receiver thread is to retrieve the incoming data sent to the instance. In practice, one of the transmission channels may not work well due to various reasons, e.g., poor wireless signal. Therefore, we design a channel switch and recovery solution to improve the reliability of our multichannel mechanism.

To reduce the transmission overhead, we combine the data fragments that can be transferred by the same transmission channel into a whole piece of data. In such way, the fragments from the same TCP payload can be transmitted for only once. Figure 6 shows the format of transmission data, which consists of a main header and several data fragments. The main header (as shown in Figure 6(b)) includes the transmission channel information and the number of data fragments.

4.4.1. Communication Instance

As each channel has its own data transmission method, there are three different communication instances for the three channels. Each communication instance is initialized by the forwarder thread corresponding to the TCP connection. In addition, the forwarder thread initializes a sender thread and a receiver thread for each channel to send and receive data, respectively.

Once data fragments are forwarded into the data transmission pool, the three sender threads can retrieve them randomly due to the randomness of the thread schedule. To improve the transmission performance, we assign different time slices to these sender threads based on their transmission speeds. In other words, the channel with higher speed could get more data. Then each sender thread combines the data fragments into a whole piece of data and adds a main header to the combined data. Later, the sender thread forwards the data to the communication instance on the remote device. While the receiver thread retrieves the incoming data sent to the communication instance and inputs the data to the data reassembling pool for processing.

4.4.2. Channel Switch and Recovery

In some scenarios, the transmission channel may not work well due to the poor wireless signal. To address this problem, we propose a channel switch and recovery solution. When the quality of service (QoS) of the data transmission channel becomes poor, our system can activate the channel switch to abandon the low-speed channel. Instead, when the QoS of the channel becomes good, the abandoned channel can be recovered. To measure the QoS of data transmission, we use the received signal strength indication (RSSI) and the round-trip time (RTT) as the indicators. When both indicators become worse, our system starts switching the channel to another one.

Figure 8 shows an example of the channel switch. Initially, the mobile device uses three channels for data transmission. For each channel, there is a corresponding transmission queue that can be used for retrieving data

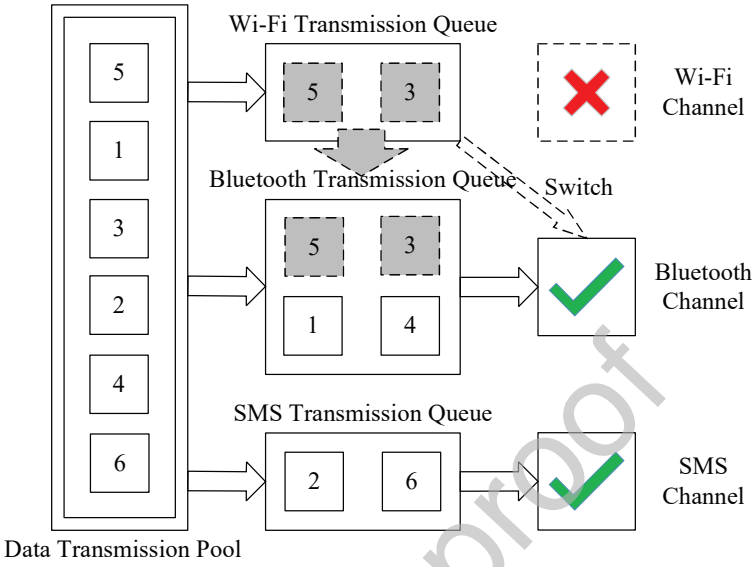


Figure 8: An example of channel switch

fragments from the data transmission pool. Specifically, there are 6 data fragments numbered from 1 to 6 in this transmission pool. Each sender thread associated with a transmission channel can obtain fragments from this pool and deliver them to the associated transmission queue. In the beginning, the Wi-Fi transmission queue will retrieve the fragments 5 and 3. When the Wi-Fi transmission channel encounters a transmission problem or the QoS of this channel becomes poor, the data transmission module will throw an exception. To deal with this exception, our method activates the transmission channel switch. In this case, the data transmission module will migrate the data fragments 5 and 3 from the Wi-Fi transmission queue to the Bluetooth transmission queue. In order to avoid the receiver application selecting the poor channel, the data transmission module will notify our MSYM to make a channel switch. In addition, the data transmission module will suspend the corresponding sender and receiver threads that are used for Wi-Fi transmission.

Figure 9 illustrates an example of channel recovery. Suppose there is a problem in the Wi-Fi transmission channel. When the problem is fixed, the transmission module can resume the sender and receiver threads of this channel. Then it can import the data fragments 5 and 3 from the Bluetooth transmission queue to the Wi-Fi transmission queue. Further, the data trans-

mission module can notify the MSYM to reuse this channel to send or receive data.

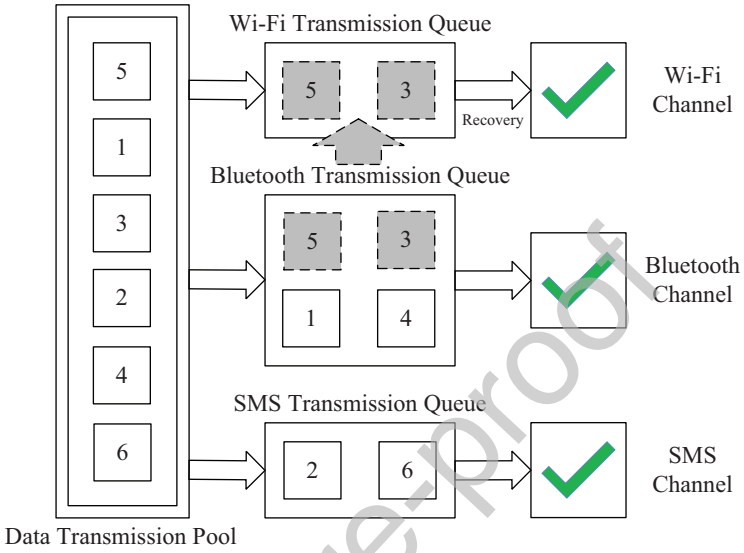


Figure 9: An example of channel recovery

4.5. Multichannel Communication Deployment

In this part, we describe how to adapt the existing communication application to leverage our MSYM for multichannel communication. *The first step* is to identify the communication port of the application. As the communication port is unique for each application, we can build an application list for multichannel communication, which can be configured manually. If the application is not in the list, MSYM can just forward their network traffic via Wi-Fi/cellular network channel directly without processing them. *The second step* involves analyzing the communication data format. Generally, the communication raw data contains headers and payload information. For the simplicity and correctness, MSYM just splits the payload (but not the headers) for multichannel communication. For the communication between MSYM and the central server, the header information should be added before splitting the data.

5. Implementation

In the evaluation, we implemented our MSYM based on Android system. The implementation relies on the VpnService to intercept network traffic and transmit data via the three transmission channels (i.e., Wi-Fi/cellular network, Bluetooth, and SMS).

5.1. Data Interception

The data interception procedure can be described as follows. Firstly, we enable the VpnService after obtaining the BIND_VPN_SERVICE permission from the Android system. Then, we set up a reader thread and a writer thread to retrieve IP packets from the VNI and inject IP packets to the VNI, respectively. To extract the header and the payload of each IP packet, we create a parser thread, which can be used to parse IP packets. Then, it creates a forwarder thread for each TCP connection aiming to maintain TCP states and process data.

To dispatch a packet to its corresponding forwarder thread, we build a mapping relationship between the forwarder thread and port numbers (both source and destination port numbers) of a packet. The reason of using the port number is due to its unique identification of TCP connection on a single device. If a packet comes from a new TCP connection, the parser thread can create a forwarder thread and bind it to the port numbers. Otherwise, the parser thread will dispatch the packet to an existing forwarder thread bound to the port numbers.

To construct the IP datagrams based on the incoming data, the forwarder thread can set up a reconstruction thread, which constructs IP datagrams and injects them to the VNI by the writer thread. To ensure the correctness and concurrency of the data delivery between multiple threads, we implemented the Lamport's ring buffer algorithm [14] based on the atomic memory operations provided by Java APIs.

5.2. Data Processing

According to the IP datagram's format, the maximum transmission unit (MTU) indicates the maximum sized datagram that can be transmitted through the network [24]. Usually, the maximum transmission unit (MTU) is set as 1500 bytes [21]. By considering the performance cost, we just split the raw TCP payload into $1 \sim 20$ fragments. It is worth noting that a larger

number of fragments means that more time consumption is required for data fragmentation and reassembling.

For the fragment header encryption, we adopted bitwise XOR operation, where we can use the same key to decrypt the encrypted header. For simplicity, we do not apply the key negotiation method to generate the shared key. Instead, we store the shared keys on the target mobile devices beforehand. To improve the data security, we implemented a simple protocol for two devices to change the shared key periodically. Specifically, we use a key array to manage the shared keys. There are 1024 keys in the key array. To refresh the shared keys, our system first utilizes the end-to-end communication channels (e.g., Bluetooth or SMS) to establish connection. Then, it notifies the two mobile devices to jointly select a different key from the key array within a pre-defined time cycle (e.g., 5 minutes). The key selection scheme and refresh time cycle can be configured. We believe that this can increase the cracking difficulty for attackers.

5.3. Data Transmission

To communicate with the remote device via the multiple channels, the forwarder thread can create three different communication instances for Wi-Fi/cellular network, Bluetooth, and SMS channels, respectively. Each channel has a sender thread and a receiver thread to either send or receive data. By considering the diverse speed among channels, we allocate different time slices for each channel aiming to achieve good performance. If one channel is given more time slice, the associated threads should get more execution time and transmit more data. As the SMS is chargeable and its transmission speed depends on a couple of factors (e.g., SMS gateway and base station), we just allocate it with a small time slice. Additionally, as the SMS channel can only support sending the string format message (e.g., ASCII and Unicode characters), it cannot directly transmit the data fragments in the binary format. To handle this issue, we leverage the Base64 encoding method to encode the outgoing binary data for the SMS channel.

For the use of Wi-Fi/cellular network channel, we utilize the SocketChannel APIs. By registering the SocketChannel with a Selector, we can use this Selector to monitor any read and write network events. When the Selector detects a read event, the receiver thread can ask the SocketChannel to read data into a buffer. Then the receiver thread can input it to the data reassembling pool. When the Selector detects a write event, the sender thread can write the data from a buffer into the SocketChannel. For the Bluetooth

channel, we use the Android Bluetooth APIs, while for the SMS channel, we use the Android SmsManager and SmsMessage APIs to send and receive short messages.

5.4. TCP/IP Stack

To maintain the TCP connection states, we implemented a TCP/IP stack according to RFC 793 [25] for our MSYM. A mobile application has to firstly establish TCP connections with MSYM, and then communicates with the remote target. A TCP connection lifecycle can be divided into three phases: connection establishment, data transfer, and connection termination.

Connection Establishment: When MSYM intercepts a SYN packet from a sender application, it should respond with a SYN_ACK packet for completing the TCP three-way handshake. When our MSYM intercepts the corresponding ACK packet from the same application, it can indicate the TCP connection between the sender application and the MSYM.

Data Transfer: When MSYM intercepts the outgoing data, it can process and transmit data. To maintain the local TCP/IP stack, MSYM should respond with an ACK packet to the sender application and update the corresponding TCP states. When MSYM intercepts the incoming data, MSYM can reassemble the data. Afterwards, it can construct the IP datagrams and then inject them into the receiver application.

Connection Termination: A TCP connection can be terminated by the sender or receiver. When MSYM intercepts a FIN packet from the sender application, it should respond with the FIN_ACK and FIN packets. After intercepting the ACK packet from the same application, MSYM can close the TCP connection and terminate the corresponding threads. Similarly, when the receiver application tries to stop the connection, MSYM can generate a FIN packet to this application. Then, MSYM would intercept the corresponding FIN_ACK and FIN packets from the receiver application. After that, MSYM can generate an ACK packet to this application. Finally, MSYM could close the TCP connection and terminate the corresponding threads.

6. Evaluation

In this section, we evaluate the effectiveness and performance overhead of our prototype implementation of MSYM. As a study, we leverage an open-source Instant Messaging (IM) application for mobile communication be-

tween two Android devices. This application employs the client-server model and has the ability to transfer plain texts and files. We deployed the server to forward network data using a laptop with one Intel 1.7GHz i5 4210U CPU and 8 GB memory. We installed the IM application and MSYM in two real Android 5.0 mobile phones[1] (including Huawei Honor 6+ Plus and Huawei Aschend Mate7).

6.1. Effectiveness

The effectiveness evaluation mainly focuses on two functionalities of MSYM. The first one is to test whether MSYM can transfer network data through multiple transmission channels. The other one is to investigate whether MSYM can perform channel switch and recovery effectively.

To evaluate the effectiveness of the multichannel transmission, we examine four combinations based on the three transmission channels. Table 1 shows all the combinations: Wi-Fi and Bluetooth channels, Wi-Fi and SMS channels, Bluetooth and SMS channels, and all of the three channels. For each combination, we configured the MSYM to open the tested channels and turn off the untested one. Then, the network data of the IM application was transferred over MYSM via the activated channels. We did not test the cellular network because our server was running on a personal computer without a public IP address. In addition, as both cellular network and Wi-Fi use the same socket APIs, we can only test the Wi-Fi or cellular network channel. To verify whether the outgoing data has been fragmented, we use the Wireshark, an open-source network packet analyzer, to intercept and analyze network packets. Also, we parse the incoming data of Bluetooth and SMS channels on the target device to verify the procedure of these two channels.

Table 1: The effectiveness of MSYM

Channel			Transmission	Channel switch	Channel recovery
Wi-Fi	Bluetooth	SMS			
•	•		✓	✓	✓
•		•	✓	✓	✓
	•	•	✓	✓	✓
•	•	•	✓	✓	✓

[1]Actually, our method can work on Android 5.0 $\sim$ 9.0 systems.

Later, we utilize the IM application to send plaintexts and files to another device, and analyze whether the target application can receive these data. As shown in Table 1, we repeated the experiments several times. It is found that the target application can receive the intact data successfully regarding the above four combinations. Furthermore, the packets captured by Wireshark indeed contain disordered and encrypted fragments, ensuring that a third-party could not obtain the raw data without knowing the data order and the decryption key.

To evaluate the channel switch and recovery mechanism, we also performed the experiments with the above four combinations. Different from the multichannel transmission evaluation, this experiment has to turn off a running transmission channel during the run time. Then, we verify whether the MSYM can switch the poor channel to another channel automatically. For this purpose, we just checked whether the IM application can work normally over MSYM. To examine the channel recovery mechanism, we first re-enable the closed channel. Then we observe the working status of the tested IM application to check if it works properly. Later, we capture the incoming data of the previously poor channel to verify whether this channel can transmit data again.

To perform the Wi-Fi channel switch evaluation, we manually shut down the wireless router in order to simulate a scenario of bad Wi-Fi signals. After the transmission channel was switched, we restarted the wireless router to test the Wi-Fi channel recovery mechanism. For testing the Bluetooth transmission channel, we move one device far away from another to weaken the Bluetooth signal. Then, we move these two devices close to each other again. Regarding the SMS transmission channel, we interrupt it by using the mobile phone jammer, which can effectively disrupt the communication between mobile phones and the base stations [37]. Similarly, we then make the SMS channel available again by stopping the jammer.

Table 1 depicts the effectiveness of channel switch and recovery mechanism. It is found that with the help of this mechanism, these two IM applications can communicate successfully over MSYM even in the bad communication environments with poor wireless signals. By capturing the transmitted data from the poor channel, it is observed that the data transmission module can reuse this channel when the QoS becomes good.

On the whole, the experimental results show that our MSYM can switch the poor channel to another one automatically, and can activate the poor channel again when its transmission state becomes normal.

6.2. Performance Overhead

To measure the performance overhead of MSYM, we utilize the open-source IM application to transfer the network data. Before validating the performance of multiple transmission, we first evaluate the performance cost for the ordinary mobile applications when MSYM was deployed with only Wi-Fi channel. Specifically, we consider two scenarios. The first scenario is to evaluate the native network performance of mobile applications without deploying the MSYM. The second scenario is to measure the network performance of mobile applications with the MSYM deployment. For this purpose, we leverage the SpeedTest, an application for testing the ping delay and network speed. To obtain the reliable results, we execute the test application ten times for each scenario and then calculate the average measurement values.

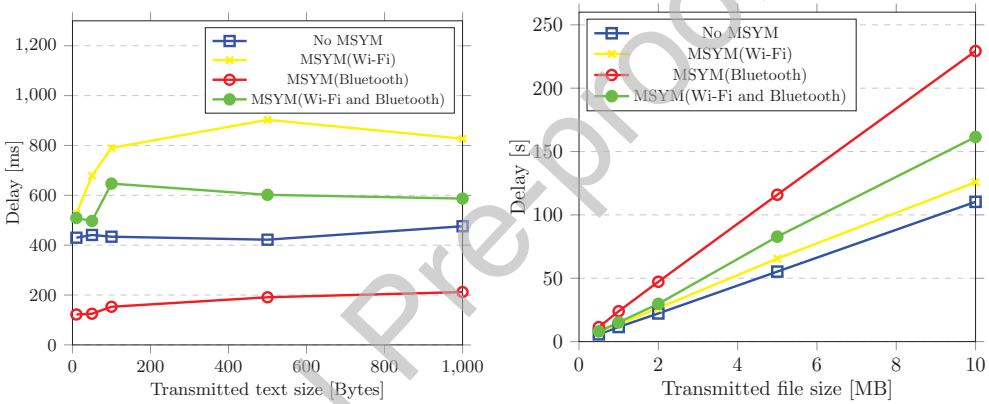
Table 2 illustrates the experiment results. Compared with the native performance, we found that the additional performance cost caused by MSYM is very small. In particular, with the deployment of MSYM, there was not ping delay, but the download speed slows down by around 6%, and the upload speed slows down by about 2.6%. The additional overhead was caused by the data interception and transmission operations of MSYM. In the data interception procedure, MSYM leverages the VpnService to intercept the outgoing packets and inject the incoming packets. MSYM needs to maintain the state of each TCP connection and uses these states to construct the IP datagrams. Also, during the data transmission, MSYM has to carry out the additional I/O operations to send and receive network data.

Table 2: The network performance by using SpeedTest.

	Ping (ms)	Download (MB/s)	Upload (MB/s)
No MSYM	2	0.89	0.39
MSYM with Wi-Fi channel	2	0.84	0.38

To test the multichannel transmission performance, we apply the IM application in four different measurement scenarios: without MSYM, MSYM with Wi-Fi channel, MSYM with Bluetooth channel, and MSYM with Wi-Fi and Bluetooth channels. As we have to perform each test several times and SMS was chargeable, we do not test the network performance of MSYM with SMS channel. In fact, the speed of SMS depends on a couple of factors (e.g., SMS gateway and base station), making it relatively difficult to measure the SMS speed.

For each measurement scenario, we use the IM application to transfer plaintexts and files between two different Android devices with various sizes. By considering the network instability, we transfer each plaintext and file ten times, and then figure out the average measurement values. Regarding the different speed of Wi-Fi and Bluetooth channels, we allocate different time slices for each channel to achieve good performance. To identify the optimal data allocation ratio, we distribute data to each channel with different ratios and then compare the network performance. After the extensive experiments, we found the best data distribution ratio to be 2:1. Accordingly, we assign the time slices for Wi-Fi channel twice than that of Bluetooth channel.



(a) Comparison of text transmission (b) Comparison of file transmission

Figure 10: Comparison of various data transmission methods

Figure 10 shows the delay of plaintext transmission (Figure 10(a)) and file transmission (Figure 10(b)) in four measurement scenarios. When transferring plaintexts with relatively small size, the Bluetooth channel incurred the lowest delay while the Wi-Fi channel introduced the highest one. In general, the transmission delay relies on two factors. The first fact is the fragmentation procedure even if MSYM uses only one single channel. The other factor is that Wi-Fi channel has to forward data to the server while the Bluetooth channel can transfer data end-to-end. However, when transferring large file, the transmission delay of Bluetooth channel increases significantly due to its relatively low speed. In particular, the transmission performance of combined two-channel (i.e., Wi-Fi and Bluetooth) is better than the single Wi-Fi channel. This is because the two-channel uses Bluetooth to transfer

partial data. For the native communication mechanism without MSYM, the transmission delay is between the Bluetooth and Wi-Fi channels. Compared with the native communication, the extra performance cost introduced by two-channel transmission is about 29% on average when transferring plaintext.

When transferring files with large size, the native communication mechanism (without MSYM) incurs the lowest transmission delay while the Bluetooth channel introduces the highest delay. When using only Wi-Fi channel to transfer the large files, the network performance is similar to the native communication. However, when using two-channel (i.e., Wi-Fi and Bluetooth) to transfer the large files, the network performance is worse than the Wi-Fi channel. The main reason is that the Wi-Fi channel has a faster transmission speed than the Bluetooth channel. Before reconstructing an IP datagram, MYSM has to ensure the split payloads transferred from Wi-Fi and Bluetooth channels are fully prepared. On the other hand, the two-channel has a better network performance than one single Bluetooth channel due to the faster transmission speed of Wi-Fi. Compared with the native communication, the additional performance cost caused by two-channel transmission is about 41% on average when transmitting files.

6.3. Power Consumption

To evaluate the power consumption of MSYM, we apply the IM application in four different scenarios: without MSYM, MSYM with Wi-Fi channel, MSYM with Bluetooth channel, and MSYM with Wi-Fi and Bluetooth channels. Since the SMS channel is chargeable, we do not test the channel. For each test scenario, we first fully charge the target Android devices. Then, we use the IM application to send and receive files between two Android devices every 10 seconds within 60 minutes.

Table 3: The power consumption rate in battery level

Scenarios	Sender	Receiver
No MSYM	3.00%	3.00%
MSYM with Wi-Fi channel	4.20%	3.80%
MSYM with Bluetooth channel	3.80%	3.70%
MSYM with Wi-Fi and Bluetooth channels	5.40%	5.00%

Table 3 shows the power consumption rate at the battery level for four test scenarios. Compared with the test case of native system, the maximal

power consumption cost of these tests is 2.4% when both of the Wi-Fi and Bluetooth channels are enabled. Particularly, the power consumption of the single Wi-Fi channel is a little bigger than the single Bluetooth channel. This result indicates the Wi-Fi channel needs more power consumption than the Bluetooth. Moreover, the send operation consumes more power than the receive operation. The results show that the fragmentation procedure needs more CPU resources than the reassembling procedure.

7. Discussion

In this work, we implemented the MSYM on Android mobile phones, but it can also be applied to another devices that adopt the Android system with the VpnService interface. In addition, our multichannel mechanism is not limited to a specific operating system. It could be applied to other mobile systems (i.e., iOS) as long as they support the VPN interface. In general, our approach can be applied to secure the End-to-End communication between mobile devices. For example, our method could be potentially used for mobile communication between unmanned aerial vehicles, whose communication is critical. To further improve the communication security, we could use a lightweight method to encrypt the TCP payload.

Since MSYM uses the Bluetooth channel for data transmission, the inherent drawback of this channel is the short communication distance. To address this problem, we can use a different transmission channel to replace the Bluetooth. For example, ZigBee based on IEEE 802.15.4 standard can be used to create personal area networks with long distances [20]. Unfortunately, many android devices do not have this equipment. In addition, some mobile devices may have two or multiple wireless network interface controllers (NICs). Thus, we could use these wireless NICs for multichannel communication.

For the communication between the server and the client, an outgoing packet usually includes some information required by a server to identify the source and target clients. Such information should be maintained during the data processing, so that the server can identify and forward this packet. Currently, MSYM only supports the communication applications when knowing their communication data format. In particular, our solution cannot apply to the communication applications that have the checksum verification mechanisms. A potential solution for this problem is to use the protocol reverse engineering [8] method to extract the message format, keeping the necessary

information and recalculating the checksum. However, MSYM can be easily applied to the mobile-to-mobile (M2M) [39] communication applications that do not require to forward the messages to the intermediate server.

8. Conclusion

The traditional mobile communication system usually uses one single channel for data transmission. To defend against the network eavesdropping attacks in such system, we design MSYM, a multichannel communication system for Android devices. In particular, MSYM first exploits the VpnService interface provided by Android system to intercept the network data. Then, it splits the intercepted network data into different fragments, and transfers these data fragments to the target mobile device via multiple transmission channels, including Wi-Fi/cellular network, Bluetooth and SMS. With the multichannel communication mechanism, it increases the cracking difficulty for attackers to recover the sensitive communication data. Our evaluation results show that our MSYM can transfer data effectively using multiple transmission channels and the performance cost is moderate.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Acknowledgement. This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 61602035, 61772078, Beijing Science and Technology Project under Grant No. Z191100007119010, and Open Found of Key Laboratory of Network Assessment Technology, Institute of Information Engineering, Chinese Academy of Sciences.

References

[1] N. J. Al Fardan and K. G. Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, May 2013.

[2] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, and François Dupressoir. Verifiable side-channel security of cryptographic implementations: Constant-time mee-cbc. In Thomas Peyrin, editor, *Fast Software Encryption*, pages 163–184, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[3] A. Alshalan, S. Pisharody, and D. Huang. A survey of mobile vpn technologies. *IEEE Communications Surveys Tutorials*, 18(2):1177–1196, Secondquarter 2016.

[4] Yu an Tan, Xinting Xu, Chen Liang, Xiaosong Zhang, Quanxin Zhang, and Yuanzhang Li. An end-to-end covert channel via packet dropout for mobile networks. *International Journal of Distributed Sensor Networks*, 14(5):1550147718779568, 2018.

[5] Matthias Bauer. New covert channels in http: Adding unwitting web browsers to anonymity sets. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, WPES '03, pages 72–78, New York, NY, USA, 2003. ACM.

[6] Birgit Bucher. Whatsapp, wechat and facebook messenger apps - global messenger usage, penetration and statistics. `https://www.messengerpeople.com/global-messenger-usage-statistics/`.

[7] Serdar Cabuk, Carla E. Brodley, and Clay Shields. IP covert timing channels: design and detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, pages 178–187, 2004.

[8] Weidong Cui, Jayanthkumar Kannan, and Helen J. Wang. Discoverer: Automatic protocol reverse engineering from network traces. In *Proceedings of the 16th USENIX Security Symposium, Boston, MA, USA, August 6-10, 2007*, 2007.

[9] Android Developers. Vpnservice. `https://developer.android.com/reference/android/net/VpnService.html`.

[10] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The secure sockets layer (SSL) protocol version 3.0. *RFC*, 6101:1–67, 2011.

[11] Steven Gianvecchio, Haining Wang, Duminda Wijesekera, and Sushil Jajodia. Model-based covert timing channels: Automated modeling and evasion. In Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection*, pages 211–230, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[12] K. Heyman. A new virtual private network for today's mobile world. *Computer*, 40(12):17–19, Dec 2007.

[13] K. J. Hole, E. Dyrnes, and P. Thorsheim. Securing wi-fi networks. *Computer*, 38(7):28–34, July 2005.

[14] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, March 1977.

[15] Butler W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613–615, 1973.

[16] Zhenhua Li, Weiwei Wang, Tianyin Xu, Xin Zhong, Xiang-Yang Li, Yunhao Liu, Christo Wilson, and Ben Y. Zhao. Exploring cross-application cellular traffic optimization with baidu trafficguard. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 61–76, Santa Clara, CA, 2016. USENIX Association.

[17] W. Liu, G. Liu, J. Zhai, Y. Dai, and D. Ghosal. Designing analog fountain timing channels: Undetectability, robustness, and model-adaptation. *IEEE Transactions on Information Forensics and Security*, 11(4):677–690, April 2016.

[18] Z. Liu, J. Chen, and T. Chen. Design and analysis of sip-based mobile vpn for real-time applications. *IEEE Transactions on Wireless Communications*, 8(11):5650–5661, November 2009.

[19] Arash Molavi Kakhki, Abbas Razaghpanah, Anke Li, Hyungjoon Koo, Rajesh Golani, David Choffnes, Phillipa Gill, and Alan Mislove. Identifying traffic differentiation in mobile networks. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, pages 239–251, New York, NY, USA, 2015. ACM.

[20] Mohammad Ali Moridi, Youhei Kawamura, Mostafa Sharifzadeh, Emmanuel Knox Chanda, Markus Wagner, and Hirokazu Okawa. Performance analysis of zigbee network topologies for underground space monitoring and communication systems. *Tunnelling and Underground Space Technology*, 71:201 – 209, 2018.

[21] D. Murray, T. Koziniec, K. Lee, and M. Dixon. Large mtus and internet performance. In *2012 IEEE 13th International Conference on High Performance Switching and Routing*, pages 82–87, June 2012.

[22] Kenneth G. Paterson and Nadhem J. AlFardan. Plaintext-recovery attacks against datagram TLS. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[23] Darren Pauli. Samsung s6 calls open to man-in-the-middle base station snooping. `https://www.theregister.co.uk/2015/11/12/mobile_pwn2own1/`. Accessed: 2015-11-12.

[24] Jon Postel. Internet protocol. *RFC*, 791:1–51, 1981.

[25] Jon Postel. Transmission control protocol. *RFC*, 793:1–91, 1981.

[26] Ashwin Rao, Arash Molavi Kakhki, Abbas Razaghpanah, Amy Tang, Shen Wang, Justine Sherry, Phillipa Gill, Arvind Krishnamurthy, Arnaud Legout, Alan Mislove, and David Choffnes. Using the middle to meddle with mobile. 2013.

[27] Abbas Razaghpanah, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman, and Vern Paxson. Haystack: In situ mobile traffic analysis in user space. *CoRR*, abs/1510.01419, 2015.

[28] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 361–374, New York, NY, USA, 2016. ACM.

[29] Eric Rescorla. The transport layer security (TLS) protocol version 1.3. *RFC*, 8446:1–160, 2018.

[30] Anastasia Shuba, Anh Le, Emmanouil Alimpertis, Minas Gjoka, and Athina Markopoulou. Antmonitor: System and applications. *CoRR*, abs/1611.04268, 2016.

[31] Shun-Chao Huang, Zong-Hua Liu, and Jyh-Cheng Chen. Sip-based mobile vpn for real-time applications. In *IEEE Wireless Communications and Networking Conference, 2005*, volume 4, pages 2318–2323 Vol. 4, March 2005.

[32] Yihang Song and Urs Hengartner. Privacyguard: A vpn-based platform to detect information leakage on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '15, pages 15–26, New York, NY, USA, 2015. ACM.

[33] Xiaoxiao Tang, Yan Lin, Daoyuan Wu, and Debin Gao. Towards dynamically monitoring android applications on non-rooted devices in the wild. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '18, pages 212–223, New York, NY, USA, 2018. ACM.

[34] A. V. Uskov. Information security of ipsec-based mobile vpn: Authentication and encryption algorithms performance. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1042–1048, June 2012.

[35] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in wpa2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1313–1328, New York, NY, USA, 2017. ACM.

[36] Wikipedia. Fisheryates shuffle. `https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle`.

[37] Wikipedia. Mobile phone jammer. `https://en.wikipedia.org/wiki/Mobile_phone_jammer`. Accessed: 2019-05-07.

[38] Daoyuan Wu, Rocky K. C. Chang, Weichao Li, Eric K. T. Cheng, and Debin Gao. Mopeye: Opportunistic monitoring of per-app mobile network performance. In *2017 USENIX Annual Technical Conference*

(USENIX ATC 17), pages 445–457, Santa Clara, CA, 2017. USENIX Association.

[39] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson. M2m: From mobile to embedded internet. *IEEE Communications Magazine*, 49(4):36–43, April 2011.

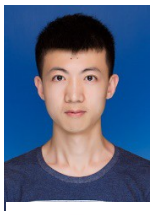
[40] Xiapu Luo, E. W. W. Chan, and R. K. C. Chang. Tcp covert timing channels: Design and detection. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 420–429, June 2008.

[41] X. Zhang, Y. Tan, C. Liang, Y. Li, and J. Li. A covert channel over volte via adjusting silence periods. *IEEE Access*, 6:9292–9302, 2018.

[42] Xiaosong Zhang, Chen Liang, Quanxin Zhang, Yuanzhang Li, Jun Zheng, and Yu an Tan. Building covert timing channels by packet rearrangement over mobile networks. *Information Sciences*, 445-446:66 – 78, 2018.

[43] Xiaosong Zhang, Liehuang Zhu, Xianmin Wang, Changyou Zhang, Hongfei Zhu, and Yu-an Tan. A packet-reordering covert channel over volte voice and video traffics. *Journal of Network and Computer Applications*, 126:29–38, 2019.

Author Biography



WENJIE WANG received the B.E. degree in software engineering from the Beijing Institute of Technology, China, in 2018, where he is currently pursuing the M.S. degree with the School of Computer Science and Technology. His research interests include software security, Android security, and vulnerability analysis.

DONGHAI TIAN received the Ph.D. degree from Beijing Institute of Technology, China, in 2012. He is a lecturer with the School of Computer Science and Technology, Beijing Institute of Technology, China. His current research interests include software security, malware analysis and detection, Android security, and cloud security.

WEIZHI MENG received the Ph.D. degree in Computer Science from the City University of Hong Kong, Hong Kong. He is an assistant professor in the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Kongens Lyngby, Denmark. His research interests include intrusion detection, biometric authentication, CPS/IoT security, malware detection, and cloud security.

XIAOQI JIA received the Ph.D. degree from Beijing Institute of Technology, China, in 2010. He is a professor at State Key Laboratory of Information Security, Institute of Information Engineering in Chinese Academy of Sciences, China. His research interests include operating system security, cloud security, and virtualization technologies.

RUNZE ZHAO received the B.E. degree in software engineering from the Harbin Engineering University, China, in 2018. He is currently pursuing the M.S. degree with the School of Computer Science and Technology in the Beijing Institute of Technology, China. His research interests include software security and machine learning.

RUI MA received the Ph.D. degree from Beijing Institute of Technology, China, in 2004. She is an associate professor with the School of Computer Science and Technology, Beijing Institute of Technology, China. Her current research interests include software security, Internet of things, neural network, and data mining.