Received September 14, 2021, accepted September 24, 2021, date of publication October 4, 2021, date of current version October 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3117337

Key Agreement Over Inter-Process Communication

MANAMI SUZUKI[1], DAI WATANABE[1], TSUTOMU MATSUMOTO[2], (Member, IEEE), NAOKI YOSHIDA[2], AND JUNICHI SAKAMOTO[2]
[1]Yokohama Laboratory, Hitachi Ltd., Kanagawa 244-0817, Japan
[2]Yokohama National University, Kanagawa 240-8501, Japan

Corresponding author: Dai Watanabe (dai.watanabe.td@hitachi.com)

ABSTRACT Today's computer is often infected by malwares and conventional communication channels such as inter-process communication (IPC) are attractive attack surface for attackers because important information such as user's personal data and passwords are transmitted between processes over IPC. In addition, there is no other protection other than the access control mechanism provided by the underlying OS, but it is not always sufficient. To improve the situation, this paper proposes a key agreement protocol between processes using a network socket, which is one of the IPC methods. Our protocol provides a means for legitimate processes to cryptographically communicate over the IPC. We use an uncertain channel for secure key agreement over IPC and we found that the IPC channel behaves as the uncertain communication channel due to the process scheduling of the OS. The proposed protocol is based on random number sharing using the messages that the attacker probabilistically fails to obtain and attacker detection who interrupts the protocol. Our protocol provides secure key sharing against an attacker that interrupts the protocol and impersonates legitimate processes. We experiment on the behavior of the uncertain channel on an actual device and confirm that our protocol achieves 128-bit security in a realistic execution time within 8.5 ms. To our best knowledge, our proposal is the first countermeasure for IPC with cryptographic strength under reasonable assumptions.

INDEX TERMS Inter-process communication, key agreement, socket hijacking, wire-tap channel.

I. INTRODUCTION

A. BACKGROUND

Today's computer systems are composed of hardware such as CPU, memory, and storage, and programs running on them. A special program called an operating system (OS) manages the entire computer so that multiple programs can share a single hardware resource without confliction. The kernel is a core program of the OS and manages the execution and termination of programs (processes) other than itself, the allocation of memory space to processes, and the order of CPU usage of processes. The memory space allocated to a process is isolated from one allocated to other processes by an access control mechanism provided by the OS. In addition, modern OSs enable event-driven execution of processes by switching the execution of processes in a short time. In this way, today's computer has a stack structure consisting of a hardware layer, an OS layer, and each layer works together to perform complex tasks.

On the OS, there are usually multiple processes running in parallel. General OSs provide the mechanisms called inter-process communication (IPC) such as shared memory, named pipe, and network socket [1]–[5]. They enable the running processes to cooperate each other to perform a single task. Multiple applications also exchange messages using IPC to perform a single task. Typical example of the latter case is a web browser application working with password management applications, music applications, and document management applications.

B. RELATED WORKS

Although IPC is used in many applications, attacks aimed at IPC have not been considered serious threats because data transmissions within a local device were considered secure. However, this is not true nowadays. Most of devices are connected to the Internet and it makes easier to break into

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Imran Tariq.

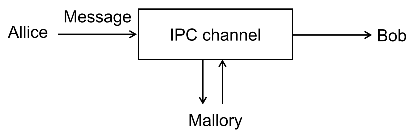


FIGURE 1. **IPC channel including attacker.**

devices by exploiting vulnerabilities. McAfee reported that COVID-19 increases the threat as the number of remote workers increases [6]. Therefore, it is becoming essential to consider the security of communications within the device in recent years.

For the attacker in the device, IPC is attractive attack surfaces because of two reasons. First, IPC is often used to exchange critical data in many applications. For example, in the password manager mentioned above, the password and personal information are exchanged over the IPC channel. Second, the security of IPC depends on access control mechanisms provided by general OSs (e.g., UNIX permission or Windows ACL) and they are not always sufficient.

We describe the problem using Figure 1. Alice and Bob are processes. Alice sends messages over an IPC channel and Bob receives them. Mallory is an active attacker and attempts to impersonate Alice or Bob by eavesdropping on the IPC channel and tampering with the messages. One may consider that the access control mechanism of OSs prevents Mallory to access to the channel, but it is not always true because the access control does not isolate a process from the other, but only isolates a user from the other, or a user group from the other. Interestingly, Android and iOS which run each application on different VMs to isolate them, provide means to exchange messages between VMs using IPC. However, Xing *et al.* reported the cases on Mac OS X and iOS that the attacker can access resources owned by other applications using IPC [7]. The paper reported the attacks on applications with client-server architecture in cases where access control does not work in IPC. Shao *et al.* reported same kind of case on Android OS [8]. Succeeding researches [8]–[12] focused on the impersonating attack to the client process (client impersonation). These attacks assume that the attacker runs with a general user privilege in the same user session to server process and client process as the legitimate process, or the attacker runs with a root privilege. The attacker succeeds the client impersonation if the attacker starts communication with the server earlier than the client. The attacker can obtain messages from the server by this attack. The attack is possible because the access control does not work as a defense mechanism and the IPC mechanism does not provide mutual authentication between the processes. Note that it is easy for the attacker to start earlier than the client in the client-server architecture, because the server always waits for a process to start communication over IPC and the client, by contrast, proceeds the communication only when needed. This fact also indicates that it is practically difficult for the attacker to start earlier than the server and impersonate it.

However, Bui *et al.* proposed in [13] a socket hijacking attack which enables the attacker to succeed the server impersonation even if the attacker starts after the server. The attack targets applications that use network sockets. The main idea of the attack is to interfere with the server's functionality by occupying the socket opened by the server and then impersonating it. The attacker can obtain data from the client by server impersonation if the attack succeeds. Since the attack is effective even if the server behaves as a daemon, the attack has expanded the range of applications that can be attacked. Their paper confirmed that the attack is effective against several real applications such as password managers.

Although the socket hijacking attack is more powerful than the previous attacks against IPC, the countermeasure is still an open problem. Bui *et al.* showed that the access control mechanism cannot be applied to a network socket in Windows, macOS, and Linux. One possible mitigation is to replace the implementation from using a network socket to other IPC methods to which Access control can be applied. However, Bui *et al.* suspect that it is practically difficult to replace because the network socket has following advantages over the other IPC methods: many OSs provide it, the functional differences between OSs are small, and the implementation of intra-process communication can be used without major modifications.. As a countermeasure at the OS layer, it is possible to use an OS to manage access control on a per-process basis (e.g., SELinux [14]). However, OSs provide such access control are not yet common, therefore countermeasures effective in common environment are needed.

In [13], Bui *et al.* stated that cryptographic technology can become the countermeasure at the application layer if legitimate processes can share a cryptographic key. Cryptographic communication between legitimate processes keeps messages secret by encrypting them and message authentication prevents impersonation because the legitimate receiver can verify the sender. The difficult point to realize this idea is to share the cryptographic key between the processes. Conventional key agreement protocols, such as PKI-based key delivery and Diffie-Hellman key exchange, achieve a session key sharing based on a public key certificate issued by a certification authority or a pre-shared secret. Unfortunately, the OS layer does not provide such a mechanism to share the key between processes. Processes within a single application may be able to share the key by using channels that can only be used between processes in a parent-child relationship (e.g., memory passing and anonymous pipe). However, the way to share the key cannot be used between processes in multiple applications (for example, between a password manager process and a web browser extension process).

While IPC has become an attractive attack target, and many attacks have been reported at the research level, the security of IPC still has many challenges, and it is desirable to establish countermeasures that can be used in general environments.

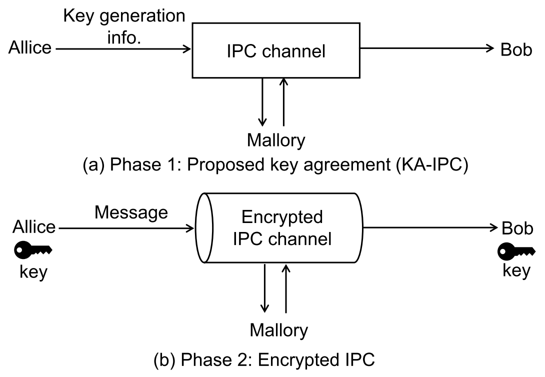


FIGURE 2. **IPC encryption using KA-IPC.**

C. CONTRIBUTION OF THIS PAPER

In this paper, we focus on encrypting the IPC channel and propose a key agreement protocol over IPC between processes, a problem to realize the IPC encryption. We call the encrypting the IPC channel IPC encryption, and the proposed key agreement protocol KA-IPC. Figure 2 shows schematic diagrams of IPC encryption using the proposed KA-IPC. Alice is one legitimate process that sends messages, and Bob is the other legitimate process that receives the messages. Mallory is an active attacker who attempts to impersonate Alice or Bob by eavesdropping on the IPC channel or tampering with the messages. Alice and Bob exchange key generation information to establish a key over the IPC channel that is not protected by technology such as cryptography or access control. After the KA-IPC, Alice and Bob encrypt the messages with the key and then transmit them over the IPC channel.

The main contribution of this paper is to disclose the way to secure the key agreement over the IPC against the socket hijacking attacker discussed in [13]. First, we find that the IPC channel can be regarded as an uncertain channel when we assume the attacker with a general user privilege. and verify this fact experimentally. Then we propose a key agreement protocol KA-IPC using the uncertain channel and evaluate its security on UDP socket. With KA-IPC, Alice and Bob can establish a secure communication channel over the IPC even though they do not have public key certificates nor pre-shared secret in advance. To our best knowledge, this is the first proposal of a key agreement protocol that focuses on the uncertainty channel with IPC. In the following, we roughly sketch their ideas.

First, we discovered the IPC channel, which has been regarded as a public channel, behaves as an uncertain channel due to a process scheduling mechanism in a general OS when processes repeatedly exchange messages over the IPC channel. We use this fact for KA-IPC. In the proposed KA-IPC, legitimate processes transmit random numbers repeatedly over the uncertain channel, and then generate a key using shared random numbers and detect the attacker. Since the attacker who tries to impersonate the server can only obtain this random number probabilistically, KA-IPC makes it probabilistically difficult for the attacker to obtain the key. KA-IPC is a countermeasure at the application layer, so it can be implemented using any IPC method other than the network socket.

As for the security of public channels such as the Internet, Shannon indicated in [15] that if an attacker can eavesdrop on all messages, i.e., if the attacker can obtain the same messages as a legitimate message receiver, the legitimate message sender and receivers cannot communicate securely without sharing a secret key in advance. On the other hand, Wyner proposed a wiretap channel model in which an attacker eavesdrops on a noisy channel [16]. Wyner's wiretap channel model assumes that the attacker cannot obtain all messages received by the receiver because of the noise. Wyner showed that when the attacker's noise is larger than the legitimate receiver's noise, it is information-theoretically possible to share messages between only legitimate entities due to the uncertainty of the noise channel, even if the legitimate sender and receiver do not share a secret key in advance. Later, Maurer proposed a key sharing method over the wiretap channel using a capability difference between legitimate entities and the attacker, which is effective even when the attacker's noise is less than the legitimate receiver's noise [17]. The proposed key sharing consists of random number exchange over a wiretap channel and information reconciliation and privacy amplification over a public channel. Such key sharing, which consists of random number exchange over an uncertain channel and information reconciliation [18], [19] and privacy amplification [18], [20] over a public channel, can be seen in many studies [21]–[23].

Though the attacker in wiretap channel model is weaker than that in Dolev-Yao model or Cannetti-Krawczyk model, it also captures the real world communication channel and many studies from both theoretical and applied aspects have been done since seminal works by Wyner and Maurer. As theoretical studies, Aggarwal *et al.* [24] and Wang *et al.* [25], [26] extended the wiretap channel model and proposed a new model that assumes active attacks such as message loss and message transmission as well as eavesdropping. In recent years, there has been a lot of applied studies, especially in the field of wireless communication channels [27]–[29].

The security of KA-IPC is based on the fact that IPC is not an ideal environment for the attacker. When multiple processes repeatedly send and receive messages over the IPC channel, the order of process execution is determined only by the process scheduler in the OS, and the order causes for the attacker to fail to receive some of the messages. Therefore, the IPC channel can be regarded as an uncertain channel. The main idea of the KA-IPC is as follows: First, the legitimate processes repeatedly exchange pairs of a random numbers and an index number as messages for key generation over this channel. As mentioned above, the attacker trying to impersonate the server probabilistically fails to obtain some of them. Then, the legitimate processes repeatedly exchange only the index numbers associated with the acquired messages over the uncertain channel. This allows the legitimate processes to know obtained random numbers by each other, i.e., they can share the random numbers. The legitimate processes use random numbers to generate a key, which the attacker cannot generate. In addition, even if the attacker sends index numbers to the legitimate processes to impersonate, the legitimate processes can detect the attacker probabilistically because the legitimate receiver receives different index numbers from both the legitimate sender and the attacker. In this way, KA-IPC can make it probabilistically difficult for the attacker, who tries to succeed in sharing the key with the client, to impersonate the server.

Contrary to PKI-based key agreements, KA-IPC is not a perfect countermeasure against arbitrary impersonating attacks. In fact, KA-IPC does not provide a cryptographic means an entity to authenticate the other process to communicate with. This is because IPC does not have functions for authentication and thus cannot inherently prevent impersonating. However, KA-IPC can reduce the success probability of the server impersonation attack to almost zero by correctly setting the number of exchanging messages as a security parameter. This paper examines server impersonating attack scenarios in KA-IPC and shows the security parameter to be secure against these attack scenarios based on experimental evaluations.

In this paper, we assume that the attacker's target is an application that consists of client-server architecture and uses a network socket as IPC. We evaluate KA-IPC under the assumption that the attacker runs in a different user session than the targets in the same manner as in [13]. However, the proposed KA-IPC can be implemented not only with a network socket but also with other IPC methods in principle and effective against the server impersonating attacks targeting other IPC methods. In addition, we can relax the assumption for the attacker's privilege. For example, KA-IPC is effective if it can be guaranteed that the IPC channel has an uncertainty for the attacker. This may be true even when the attacker runs in the same user session as the targets, or the attacker has a root privilege. Note that KA-IPC is not effective against an attacker with powerful attack capabilities such as forcibly rewriting the execution orders of processes. In this paper, we exclude such cases to simplify the discussion.

D. ORGANIZATION

The rest of this paper is organized as follows: Section II introduces terms and attacks that we focus on. Section III focuses on properties of the OS and the network socket that are essential to KA-IPC, and Section IV defines our attack model. Section V proposes the KA-IPC protocol. Section VI discusses the attack scenarios. Section VII experimentally characterizes the communication channel used in KA-IPC and shows the recommended security parameters in terms of security. Finally, Section VIII summarizes this paper.

II. PRELIMINARY

A. NETWORK SOCKETS AS IPC

This section describes network sockets as IPC. We explain it using an example of an application with the client-server structure. The client and server create sockets using socket information consisting of an IP address and a port, and they must share the socket information before starting communication via the network socket. The server creates a socket bound to socket information and then waits for the client to connect to the socket. If the server uses the loopback interface, i.e., localhost addresses 127.0.0.0/8 or ::1/128 when the server creates the socket, only processes running on the same OS can connect to the server's socket. The client also creates a socket bound to the same socket information, connects to the server's socket, and exchanges messages with the server.

The network sockets mainly use TCP or UDP as a transport layer protocol. We call network sockets using TCP TCP sockets and one using UDP UDP sockets.

In the TCP socket communication, two processes establish the connection and communicate one-to-one. Another process cannot interrupt and communicate with these processes after the connection has been established. In addition, the received message is stored in a receive buffer associated with the receiver's socket, and the OS requests resending of a lost message when the OS finds the lost message by confirming the received buffer. Therefore, there is no message loss in TCP connection in principle, i.e., all messages sent by the sender are received by the receiver. Because of these features, the TCP socket communication used within many applications. However, if a process already takes a port, another process cannot use this port to create the socket. To avoid a process cannot communicate by the socket collision, the process using the TCP socket often has more than one port candidate.

On the other hand, UDP uses a connection-less communication. At the beginning of the communication, the receiver goes into a ''waiting'' state. The receiver only can receive the message sent during the waiting state. After receiving a message, the receiver is automatically released from the waiting state. If the receiver wants to receive the message again, the receiver needs to go into the waiting state again. Because the receiver can receive messages only when it is in the waiting state, in the UDP socket communication, the sent messages may not be received by the receiver. Therefore, the UDP socket channel can be regarded as a communication channel where message loss occurs. Since no process can basically occupy a port all the time, each process is not necessary to have several port candidates as TCP communication. Another important property of UDP socket is that when multiple receivers receive a message using the same socket, the receiver who accesses the socket first occupies the socket. Another receiver can receive a message using the same socket after the socket is released. Note that a message is received by only one receiver.

B. POTENTIAL VULNERABILITIES OF NETWORK SOCKETS AS IPC

This section describes potential vulnerabilities of network sockets as IPC. Any process that knows the port of the target socket can send to the target socket and receive messages to the target socket by creating a socket with the same port as the target. The port can be easily obtained even by a non-legitimate process through static analysis [13], [30] (e.g., examining an application documentation and a source code) or dynamic analysis (e.g., using a system call ''netstat'' and a port scan).

One thinks that an access control mechanism and authentication mechanism are naïve countermeasures. However, the network socket as IPC provides neither. Another possible countermeasure could be to establish an authentication mechanism for processes at the application layer. For that, processes must have a public key certificate issued by a certification authority or pre-shared secret, but the OS does not provide the mechanisms for them.

For these reasons, the attacker can easily impersonate legitimate processes. Many applications adopt TCP socket more than UDP socket because of the reliable communication. However, in such applications, an attacker once establishes the connection with a legitimate process before another legitimate process does; the attacker succeeds receiving and sending all messages between the process with the connection. If the applications have a client-server architecture, the attacker can impersonate the client process easily because the client runs only when necessary. On the other hand, most of the server process behaves as a daemon. In such an application, it is difficult for the attacker to start earlier than the server, i.e., the attacker cannot impersonate the server easily.

C. SOCKET HIJACKING IN NETWORK SOCKETS

Bui *et al.* proposed a socket hijacking attack that allows the attacker to impersonate the server even if the attacker starts after the server [13]. The attack can be a threat to many applications because the attack is effective even against applications where the server runs as a daemon. In this attack, the attacker interrupts the protocol and impersonates the server by taking advantage of the fact that it occupies the socket after the TCP communication establishment and that the legitimate processes have multiple candidate ports. In this paper, we focus on the attack and describe the details below.

First, we describe the attack model assumed in [13]. The attacker's goal is to obtain the client's data by impersonating the server. The attacker's target is the application that has the client-server structure and uses the TCP sockets. The client and server have at least two port candidates (a primary and secondary port). The attacker is a non-privileged process. The server and client run in a same user session and the attacker runs in a different user session. In addition, the attacker does not start before the server.

Next, we explain the way that the attacker impersonates the server by the socket hijacking. At the beginning, the server starts and then creates a socket with the primary port and waits for a connection from the client. The attacker starts after the server and then connects to the primary port where the server is waiting. The attacker establishes the connection with the server. The attacker also creates a socket with the secondary port and waits for a connection from the client. After that, the client starts and then tries to connect to the primary port, but the client cannot because the attacker has taken the primary port. Therefore, the client connects to the secondary port where the attacker is waiting to connect. In this way, the connection between the attacker and client is established, and the attacker can impersonate the server. In [13], Bui *et al.* reported that the server impersonating attack is effective in real applications (e.g., a password manager and music streaming service) to obtain important information (e.g., user information).

III. FUNCTIONS OF UNDERLYING OS AND CHANNELS REQUIRED FOR KA-IPC

We assume the underlying OS employs preemptive multitasking as same as general OSs such as Linux, macOS and Windows. KA-IPC uses two communication channels, an uncertain channel and a multiple sender detecting channel, that can be constructed by IPC, taking advantage of features of preemptive multitasking. Section III-A describes the features of preemptive multitasking to construct communication channels for KA-IPC. Sections III-B and III-C describe the way to construct the uncertain channel and multiple senders detecting channel using IPC, respectively.

A. PREEMPTIVE MULTITASKING

Most of the current OSs run multiple processes in parallel. There are two mechanisms for this: cooperative multitasking and preemptive multitasking. General OSs such as Linux, macOS, and Windows employ the latter in which the OS manages the executions and interrupts of running processes. For preemptive multitasking, the OS uses the CPU's interrupt functions to switch between executing and stopping processes in a short time on the CPU. The execution order of processes is controlled by a process scheduler built into the OS kernel. The scheduler periodically determines the order of execution of processes using CPU information such as a hardware timer, process priorities, and CPU utilizations.

Note that in preemptive multitasking, when processes repeatedly access hardware and software resources, the order in which the processes access the resources varies. For example, in the UDP socket communication, we consider the case where multiple senders repeatedly send messages via a fixed UDP socket channel and a receiver receives them. In this case, the order of sent messages means the order of access to the socket by the senders, which is determined only by the OS. Therefore, each sender cannot know the order in which the messages were sent and whether the sent messages were received. In addition, the receiver cannot identify the sender from the received message.

B. COMMUNICATION CHANNEL WITH UNCERTAINTY

Here we describe the way to construct an uncertain communication channel with UDP sockets. The uncertainty is realized by taking advantage of preemptive multitasking. Let us consider the following communication using the UDP sockets: The sender repeatedly sends different messages each time without disconnecting the socket. The receiver repeats to disconnect and reconnects to the socket for each message reception, not to occupy the socket for a long time. Under this setting, the receiver can receive only messages sent when the receiver is in the waiting state and the execution order of the process is receiver-then-sender because of the characteristic of UDP socket. In addition, neither the sender nor the receiver can know which messages are sent or received by the other. It is because the execution order of processes is determined only by the OS, and the order information is stored in the kernel memory. Therefore, in this case, the communication channel can be regarded as a channel where the receiver receives the message probabilistically, i.e., the communication channel with uncertainty. This is true even when multiple processes, including the attacker, are sending, or receiving messages. We call the channel IPC1. Note that it is also possible to construct IPC1 using other IPC mechanisms in the same way.

C. COMMUNICATION CHANNEL THAT PROBABILISTICALLY DETECTS MULTIPLE SENDERS

We describe the way to construct a communication channel that probabilistically detects multiple senders with UDP sockets. The channel can be realized by taking advantage of preemptive multitasking, just like IPC1. Let us consider the following communication using the UDP sockets: The sender and receiver repeatedly send and receive messages. The difference from IPC1 is that the sender always sends the same message. Due to features of preemptive multitasking when multiple senders repeatedly send messages on the same socket, the receiver probabilistically receives the message from each sender. The higher the number of times the receiver receives the message, the higher the probability that the receiver receives at least one message from each sender. If multiple senders send messages with different values, the receiver can probabilistically detect multiple senders. Therefore, we can regard this communication channel as a channel that the receiver can probabilistically detect multiple senders. We call the channel IPC2. Note that it is also possible to construct IPC2 using other IPC mechanisms in the same way.

IV. ATTACK MODEL

In this section, we describe the assumed environment and model the communication channel used for KA-IPC, which is proposed in Section V. We also describe an assumed attack model in this paper, which is the same as in [13]. According to the attack model, we organize attack scenarios on KA-IPC and theoretically calculate attack success probabilities in each scenario in Section VI, and experimentally determine the probabilities in Section VII.

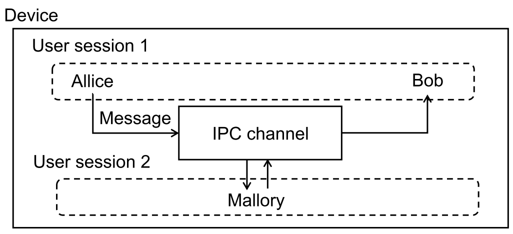


FIGURE 3. Assumed environment and entities.

A. SECURITY GOAL

Current cryptography is considered secure for general use if the computational complexity of any attack is at least 2^{128}. This is simply called "the security level is at least 128 bits." We discuss the security in terms of the attack success probability. In this paper, we consider that KA-IPC achieves 128-bit security when the attack success probability is less than 2^{-128} for all possible attack scenarios under the attack model described in this section. We describe the details of the attack scenarios in Section VI.

B. ASSUMED ENVIRONMENT

In the following, we describe the communication channel as the abstraction of UDP socket. In Figure 3, among the legitimate processes, Alice plays as the sender process and Bob as the receiver process. We assume that Mallory plays as an active attacker, who exchanges messages with the client to impersonate the server. Mallory runs with a non-privileged user in a different user session than Alice and Bob. Mallory only does what a process with a non-privileged user running on the OS can do, i.e., Mallory cannot access memory spaces used by other processes. Mallory creates a socket with the same socket information as Alice and Bob. Note that Mallory cannot eavesdrop directly on the UDP socket channel because of characteristic of the UDP socket channel. Instead, Mallory can receive a message sent by Alice to Bob, and then resend the received message to Bob. It can be regarded as eavesdropping, so we call this behavior eavesdropping in this paper. Mallory can also behave tampering with a message by sending a different message after receiving a message like eavesdropping. However, tampering is not an effective for the server hijacking in KA-IPC, so we consider the attacker who receives, sends (inserts), and eavesdrops. We assume that the attacker does not start earlier than the server. It is a realistic assumption, assuming that the server is running as a daemon.

C. COMMUNICATION CHANNEL USED IN KA-IPC

1) MODELING COMMUNICATION CHANNEL

In Section III, we explained that the UDP socket channel could be regarded as IPC1 (Communication channel with

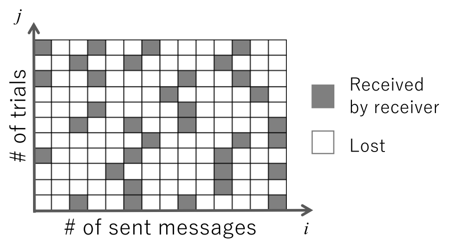


FIGURE 4. **Sent data by sender.**

uncertainty) and IPC2 (Communication channel that probabilistically detects multiple senders) when the receiver and sender transmit messages repeatedly. The only difference between IPC1 and IPC2 is the content of messages, and the essential nature of these channels is the same: both these channels take advantage of the uncertainty of the execution order of processes. In the following, we model the uncertain communication channel.

The UDP socket channel between the sender Alice and the receiver Bob is denoted by $\mathcal{C}(A, B)$. We divide the time by an execution of Bob and call the interval a period. Alice sends $n(t)$ messages during a period t and they are denoted by $x_i(t)$ $(1 \leq i \leq n(t))$. Bob certainly receives one of the messages $x_i(t)$ during the period t. The probability of Bob's receiving the i-th message $x_i(t)$ in the period is denoted by

$$\{p_{x_i(t)}^{\mathcal{C}(A,B)} = prob^{\mathcal{C}(A,B)}(A \to_{x_i(t)} B)\}_i.$$

2) VALIDITY OF OUR CHANNEL MODEL

The model in Section IV-C1 implicitly assumes that Alice never occupies the communication channel over the long time and Bob receives a message within a certain time period. In this paper, we experimentally confirm that this assumption holds on Linux. We show the environment used in the experiment below.

- CPU: BROADCOM BCM2837 ARMv8 4core 1.2GHz
- OS: Raspbian 4.9.2 64bit
- RAM: 1GB
- Language: C
- Compiler: gcc 4.9.2

We used Raspberry Pi3 model B [31] for hardware and software. We explain this reason in Section VII-A.

We run two processes, a sender and a receiver in a same user session. The sender sends 1, 2, 3, ... as messages and the receiver iterates to receive messages. In the experiment, the receiver receives 1,000 messages in a trial, and we execute 100 trials. Figure 4 picks up a part of experimental results. It does not only support the assumption, but also shows that the UDP socket is definitely uncertain.

D. ATTACKER'S ABILITY

1) BASIC ACTIONS

This section describes what the attacker Mallory can do. As described in Section IV-B, Mallory can access the channel and send and receive messages, like Alice and Bob.

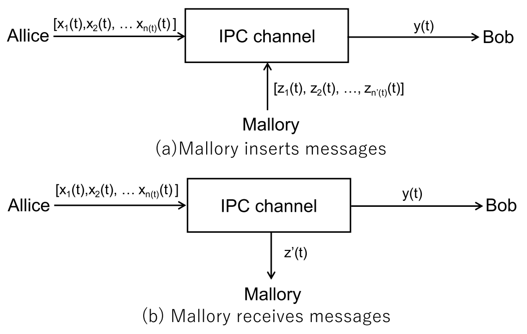

Allice
[x₁(t),x₂(t), ... xₙ₍ₜ₎(t)]
IPC channel
y(t)
Bob
[z₁(t), z₂(t), ..., zₙ'₍ₜ₎(t)]
Mallory
(a)Mallory inserts messages
Allice
[x₁(t),x₂(t), ... xₙ₍ₜ₎(t)]
IPC channel
y(t)
Bob
z'(t)
Mallory
(b) Mallory receives messages

FIGURE 5. **Attacker's action.**

We denote the channel between the sender Alice and the receiver Bob with Mallory by $C(A, B; M)$. Here, Mallory is considered as a noise factor in the channel $C(A, B)$. In the consideration of success probability of attacks, we also use the notations $C(A, M; B)$ and $C(M, B; A)$ to specify the correspondence between the entities and the rolls. For example, we use $C(A, M; B)$ when we consider the event that the messages sent by Alice are not received by Bob, but by Mallory.

Figure 5 (a) shows the schematic of the communication channel when Mallory acts as Alice. Mallory sends $n'(t)$ messages, $z_j(t)$ $(1 \leq j \leq n'(t))$, during the period t, and at this time, Alice also sends $n(t)$ messages $x_i(t)$ $(1 \leq i \leq n(t))$. Bob receives a message $y(t)$ sent by either Alice or Mallory in period t, i.e., $x_i(t)$ or $z_j(t)$. Let $P_{M(s)}[A]$ and $P[M(s)]$ be the probabilities that Bob receives a message from Alice and Mallory during t, respectively. They are given by

$$P_{M(s)}[A] = \sum_i p^{C(A,B;M)}(x_i(t)),$$

$$P[M(s)] = \sum_j p^{C(M,B;A)}(z_j(t)),$$

where $M(s)$ means Mallory acting as a sender. We omit the notation if the roll of Mallory is obvious from the context Then the following relationship holds: $P[A] + P[M(s)] = 1$. $P[A]$ and $P[M(s)]$ can be calculated by the sum of the probabilities that Bob receives the message $x_i(t)$ sent by Alice and the message $z_j(t)$ sent by Mallory, respectively. Figure 5 (b) shows the schematic of the communication channel when Mallory acts as Bob. Mallory receives message $z'_{(}t) = x'_i(t)$ $(1 \leq i' \leq n(t))$ according to the probability distribution $p_{x'_i(t)}^{C(A,M;B)}(t)$ in the same manner as in Section IV-C. Note that the period t is defined as the execution of Bob, therefore Mallory may not receive any message or may receive more than a message in the period t. In addition, one message is received by only one receiver, therefore messages received by Bob and Mallory are always different in this channel.

2) SPECIFIC ATTACK METHODS

Here, we describe the specific attack methods by Mallory. As mentioned in Section IV-B, Mallory can eavesdrop, which is equal to "receive then resend," insert (send), and receive messages. We denote their success probabilities by p_{eve}, $p_{ins} = P[M(s)]$, and $p_{recv} = P[A]$, respectively. If Bob receives the resent message, it can be considered as the success of eavesdropping. Thus, p_{eve} is given by $\sum_{i=1}^{n} prob^{\mathcal{C}(A,B;M)}(A \to_{x_i(t)} M) \cdot prob^{\mathcal{C}(A,B;M)}(M \to_{x_i(t)} B|A \to_{x_i(t)} M)$. To simplify the discussion, we consider eavesdropping as a one of message sending. Let $P[A]$ and $P[M(e)]$ be the probabilities that Bob receives a message from Alice and Mallory during t, respectively. The following relationship holds: $P[A] + P[M(e)] = 1$. $P[A]$ and $P[M(e)]$ are the sum of the probabilities that Bob receives the message $x_i(t)$ sent by Alice and the message $x_i'(t)$ resent to eavesdrop by Mallory, respectively.

3) CONSTRAINTS ON ATTACKS

As described in Section IV-B, an attacker cannot access information stored in the memory allocated to each process such as intermediate and resulting values of encryption or decryption function during cryptographic communication after KA-IPC, and a cryptographic key generated in KA-IPC. It is because processes with a non-privileged user, including the attacker, cannot access the memory space used by other processes. In addition, we do not assume message leakage by side-channel attacks such as timing attacks and row-hammer attacks as in [13]. We do not either assume that DoS attacks interfere with the protocol execution since the attacker's goal is to obtain the client's data by impersonating the server. We also assume that the attacker starts after the server, like the assumption in the socket hijacking attack [13]. The attacker interrupts the protocol and impersonates the server by inserting messages or eavesdropping, as described in Section IV-D2.

E. RELATIONSHIP WITH CONVENTIONAL COMMUNICATION CHANNEL MODELS

We explain the differences between our channel model and the conventional one. Although there is no physical noise on the UDP socket channel, the channel is regarded as a channel where a message probabilistically is lost as described in Section II-A. A remarkable difference between UDP socket and precedent studies is that a message is received by only one receiver when there are multiple receivers. Therefore, our communication channel model is neither compatible with the wiretap channel model [16] where the attacker only eavesdrops nor the broadcast model [17] where multiple entities probabilistically receive broadcasted messages.

V. KEY AGREEMENT OVER IPC CHANNEL

We propose a key agreement protocol over an IPC channel, KA-IPC. This paper describes KA-IPC over a UDP socket channel; note that the socket hijacking [13] does not occur in the UDP socket communication. KA-IPC enables cryptographic communication over a TCP socket channel (or a UDP socket channel) using a key, and it enables IPC between processes while keeping messages secret from the attacker

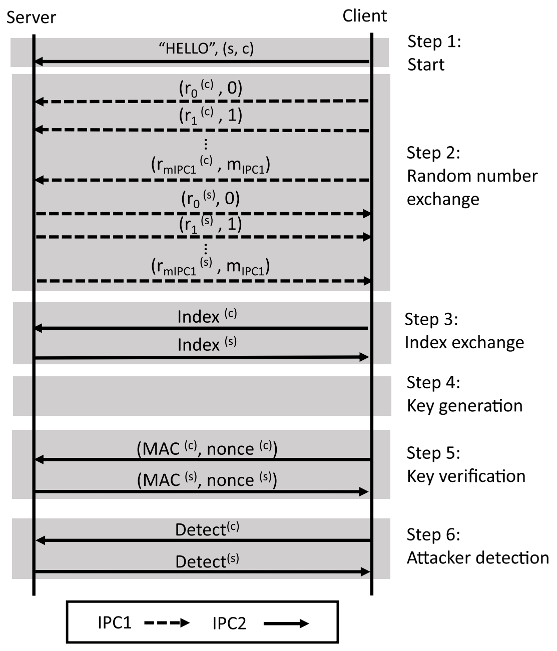


FIGURE 6. Protocol diagram of proposed key agreement (KA-IPC).

trying to impersonate the server. This section describes the basic concept of KA-IPC and then explains details of each step of KA-IPC.

A. BASIC CONCEPT

KA-IPC uses two channels, the uncertain communication channel IPC1 and the channel that can detect multiple senders IPC2. These channels realizes that the client and server share random numbers to generate a key and detect the attacker. KA-IPC make it probabilistically difficult for an attacker to share a key with the client for the server impersonation. KA-IPC has security parameters, the numbers of sent and received messages, which can be increased to reduce the success probability of the server impersonation attack.

Figure 6 shows a protocol diagram of KA-IPC. KA-IPC consists of the following six steps:

- Step 1: Start
- Step 2: Random number exchange
- Step 3: Index exchange
- Step 4: Key generation
- Step 5: Key verification
- Step 6: Attacker detection

In the following, we describe each step in detail. We denote the numbers of receiving messages over the IPC1 channel and IPC2 channel as m_{IPC1} and m_{IPC2}, respectively, and similarly, the number of sending messages over the IPC1 channel and IPC2 channel as $n_{IPC1} = n(t) \cdot m_{IPC1}$ and $n_{IPC2} = n(t) \cdot m_{IPC2}$, respectively. Here, $n(t)$ is the number of sent messages during t. These numbers of sending and receiving messages are determined as the specification of the protocol in advance.

B. STEP 1: START

After starting, the server waits until it receives a start signal "*HELLO*", (s, c) over the IPC2 channel. The client starts and then sends the start signal.

C. STEP 2: RANDOM NUMBER EXCHANGE

The client and server each repeatedly send and receive a pair of a random number and index over the IPC1 channel. The client sends a pair of a random number $r_i^{(c)}$ and index i, $(r_i^{(c)}, i)$, similarly, the client sends a pair of a random number $r_i^{(s)}$ and index i, $(r_i^{(s)}, i)$. In this step, the client and server each receive m_{IPC1} random numbers. If the client and server cannot receive m_{IPC1} random numbers, they sent an attacker detection signal in Step 6.

D. STEP 3: INDEX EXCHANGE

The client and server each repeatedly send and receive indexes received in Step 2 as index information $Index^{(c)}$, $Index^{(s)}$ over the IPC2 channel, respectively. The client and server send an attacker detection signal in Step 6 if they receive different values more than once or cannot receive m_{IPC2} index information in this step. The client and server can identify random numbers associated with received index information, i.e., they can know random numbers that were successfully shared in Step 2. The client and server share $2m_{IPC1}$ random numbers through Steps 2 and 3.

E. STEP 4: KEY GENERATION

The client and server each generate a cryptographic key from a key derivation function (KDF) using $2m_{IPC1}$ random numbers shared between them. For example, HKDF based on hash-based message authentication code (HMAC) [32] can be used for KDF. When the client and server successfully share random numbers in Steps 2 and 3, the keys generated by each client and server have the same value.

F. STEP 5: KEY VERIFICATION

The client and server each exchange a message authentication code (MAC) value calculated using the key generated in Step 4 over the IPC2 channel and check whether they succeed in generating the same key. In this step, they detect the attacker in the same way as in Step 3. The client and server repeatedly send and receive a pair of nonce, $nonce^{(c)}$ and $nonce^{(s)}$, and calculated MAC value $MAC^{(c)} = MAC(nonce^{(c)}, key)$ and $MAC^{(s)} = MAC(nonce^{(s)}, key)$, respectively, i.e., they exchange MAC value information $(MAC^{(c)}, nonce^{(c)})$ and $(MAC^{(s)}, nonce^{(s)})$. After receiving MAC value information, the client and server calculate a MAC value from received nonce and own key, and check whether the MAC value is the same as the received MAC value. If the client or server finds that any of the following is true, the client or server detects an attacker and sends an attacker detection signal in Step 6.

- The values of received m_{IPC2} MAC value information are not all the same.
- The received MAC value and the calculated MAC value are different.
- m_{IPC2} MAC value information are received.

G. STEP 6: ATTACKER DETECTION

The client and server each repeatedly send and receive attacker detection signals $Detect^{(c)}$ and $Detect^{(s)}$ over the IPC2 channel. They send the attacker detection signal to inform whether an attacker is detected in Step 3 and Step 5. The client and server terminate KA-IPC as the key sharing failed when they detect the attack.

VI. SECURITY OF KA-IPC

In this section, we discuss the security against the server impersonation attack in KA-IPC and estimate the attack success probability. we first define what success in the server impersonation in KA-IPC is. Then, we describe possible attack scenarios based on the attack model defined in Section IV and explain their success probabilities. We do not describe trivial attacks such as brute force attack on the shared key and focus on most likely scenarios for simple discussion. We neither consider the attacks using vulnerabilities outside of KA-IPC, such as to obtain a seed for random number generation by privilege escalation attacks, or similarly to obtain the client's key by side-channel attacks. As described in Sections IV-C1 and IV-C2, the attacker can eavesdrop, receive, and insert messages. In the following, we use "obtain" to indicate that the attacker uses these actions to obtain a specific message.

A. DEFINITION OF SUCCESSFUL SERVER IMPERSONATION IN KA-IPC

We define the success of the server impersonation attack as the attacker generating the same key as the client's key without the client detecting the attacker in KA-IPC. The attacker must satisfy following two conditions to success the attack.

Condition 1: Random number sharing

The attacker obtains random numbers used by the client for key generation in Step 2 (Random number exchange) and identifies exactly those used for key generation in Step 3 (Index exchange).

Condition 2: Detection avoidance

The client does not detect the attacker in Steps 3, 5 (Key verification), and 6 (Attacker detection).

Note that even if the server detects the attacker, the server impersonation attack succeeds if the client does not obtain the attacker detection signal from the server. We describe the details of this case in Section VI-B2.

B. ATTACK SCENARIOS

Next, we investigate the attacker's possible actions concerning the conditions described in Section VI-A and the resulting client's and server's states. Sections VI-B1 and Section VI-B1 describe details about Condition 1 and

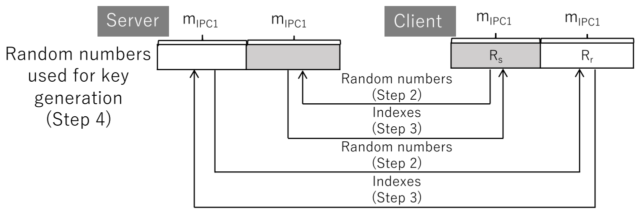


FIGURE 7. Flow of information used for key generation.

Condition 2, respectively. Section VI-B3 describes details of attack scenarios as their compositions.

1) RANDOM NUMBER SHARING

Here, we describe the way that the attacker obtains random numbers, which are used to generate the client's key (Condition 1). First, we explain the random numbers used by the client for key generation. Figure 7 shows the flow of information related to key generation when KA-IPC is performed only by the client and server. To share the random numbers between the client and server, it is necessary to exchange the random number information in Step 2 (Random number exchange) and the index information in Step 3 (Index exchange). In Fig. 7, the same color means the same value, and arrows means show the flow of information in Step 2 and Step 3. We focus on the client's key. The key is generated by $2m_{IPC1}$ random numbers. Half of them, m_{IPC1} random numbers, are received in Step 2. The remaining are a subset of random numbers sent by the client in Step 2 and then identified using received index information in Step 3. Let R_r be the random numbers received in Step 2, and R_s be the remaining.

Next, we explain the way for the attacker to share all random numbers used for a key generation with the client, i.e., to obtain R_r and R_s. In the following, we do not consider whether the attacker shares random numbers with the server. As described in Section IV-C1 and Section IV-C2, the attacker can receive, eavesdrop, and insert messages in Step 2 and Step 3 to obtain R_r and R_s with the client. Here, the messages mean random number information in Step 2 and index information in Step 3.

First, we describe the attacker's actions and the client's and server's states caused by the attacker's actions to obtain R_r and R_s in Step 2.

Obtaining R_r:

The attacker only can "eavesdrop" and "insert" random number information to obtain R_r. This paper only considers that the attacker performs either "eavesdropping" or "insertion" in an attack scenario, but not both. For each action, the following Cases (1-1) or (1-2) must be satisfied.

(1-1): The attacker "eavesdrops." The attacker obtains at least m_{IPC1} random number information sent by the server, and the client obtains m_{IPC1} random number information eavesdropped by the attacker.

(1-2): The attacker "inserts" random number information. The client obtains m_{IPC1} random number information from the attacker.

TABLE 1. Attacker's action in Step 2.

Case	Obtaining R_r		Obtaining R_s	
	(1-1) Eavesdrop	(1-2) Insert	(2-1) Eavesdrop	(2-2) Receive
(a)	✓		✓	
(b)		✓	✓	
(c)	✓			✓
(d)		✓		✓

Obtaining R_s:

The attacker must obtain at least m_{IPC1} random number information sent by the client in Step 2 to obtain R_s. The attacker can "eavesdrop" or "receive" to obtain random number information. For each action, the attacker must satisfy the following Cases (2-1) or (2-2).

(2-1): The attacker "eavesdrops." The attacker obtain at least m_{IPC1} random number information sent by the client, and the server obtains m_{IPC1} from the random number information eavesdropped by the attacker.

(2-2): The attacker "receives" m_{IPC1} random number information sent by the client.

Table 1 shows these attacker's actions in Step 2 for the attacker to obtain R_r and R_s. The attacker must perform either of Cases (1-1) and (1-2), and either of Cases (2-1) and (2-2) in Step 2. Therefore, the attacker's actions in Step 2 can be classified into Cases (a) - (d).

Next, we describe the attacker's actions, and the client's and server's states caused by the attacker's actions necessary to obtain R_r and R_s in Step 3. The attacker decides the next action in Step 3 under the assumption that he succeeds on the action in Step 2. In the following, we describe the attacker's actions in Step 3 and the client's and server's states caused by the attacker's action for each action case in Step 2.

Obtaining R_r:

(1-1): The attacker "receives" index information sent by the client once.

Since the attacker assumes to have obtained the random number information, including the one obtained by the client in Step 2, the attacker tries to obtain the index information sent by the client to identify R_r. Unlike the client and server, the attacker does not need to perform attacker detection, so it is sufficient if the attacker obtains the index information once. The possible attacker's actions to obtain the index information are "eavesdropping" and "receiving." Due to the characteristics of IPC2, even if the attacker "receives" one index information, the server can obtain m_{IPC2} index information as if there was no attacker. In the following, we do not consider that the attacker "inserts" an obtained message (i.e., "eavesdrops"), but only consider the case of the attacker "receives."

(1-2): The attacker repeatedly "inserts" index information associated with the random number obtained from the client in Step 2, and the client obtains m_{IPC2} index information sent by the attacker.

TABLE 2. Possibility of detecting attacker.

Case	(3-1)	(3-2)	(3-3)	(3-4)
(a)				
(b)		✓	✓	✓
(c)	✓	✓	✓	✓
(d)	✓	✓	✓	✓

R_r is the random numbers associated with the index information in Step 3, among the random number information received by the client in Step 2. Therefore, the attacker must repeatedly insert index information, and the client must obtain m_{IPC2} index information from the attacker for the avoidance of the attacker detection by the client. Otherwise, the client receives at least index information from the server so that the client detects the attacker. We discuss the avoidance of the attacker detection in detail in Section VI-B2.

Obtaining R_s:

(2-1), (2-2): The attacker "receives" index information sent by the server once.

It is the same as Case (1-1) when the attacker obtains R_r.

In the above, we have discussed the cases that the attacker shares random numbers with the client without considering that the attacker shares random numbers with the server, but in Case (a), the attacker shares random numbers with both.

2) DETECTION AVOIDANCE

One of the conditions for the attacker to generate the same key as the client's key is that the attacker avoids attacker detections by the client (Condition 2). Here, we assume that Condition 1 is satisfied and describe the way the attacker avoids the detections in Steps 3, 5, and 6. The following events are possible for the client to detect the attacker in KA-IPC.

(3-1): The client receives multiple index information with different values in Step 3 (Index exchange).

(3-2): The client receives multiple MAC values with different values in Step 5 (Key verification).

(3-3): The received MAC value does not equal to that calculated by the client in Step 5 (Key verification).

(3-4): The client receives the attacker detection signal from the server in Step 6 (Attacker detection).

Table 2 shows the possibility of detecting the attacker by these events in each case. Case (3-1) occurs probabilistically when the attacker "inserts" index information in Step 3, i.e., in Cases (b) and (d). In addition, Cases (3-2) - (3-4) are probability occur when the client and server share different random numbers, i.e., in Cases (b) - (d). Therefore, the attacker is probabilistically detected by the client, excluding in Case (a).

Next, we describe the attacker's actions to prevent to be detected by the client, and the client's and server's states caused by the attacker's actions. Case (3-2) is the event that the client receives both the MAC information sent by the server and attacker, and Case (3-3) is the event that the client receives only the MAC information sent by the server. In addition, Case (3-4) is the event that the client receives the attacker detection signal sent by the server. Therefore, the necessary and sufficient condition for the attacker not to be detected is that the client does not receive any of the messages sent by the server in Steps 3, 5 and 6. To achieve this, the attacker must perform and succeed in at least one action among the following two actions: the first one is that the attacker repeatedly obtains messages sent by the server to prevent the client to receive any message from the server until the server stops to send messages. Then the attacker repeatedly inserts a message over the IPC2, and the client receives the m_{IPC2} messages from the attacker. The second is that the attacker starts the steps earlier than the server and repeatedly inserts a message on IPC2, and the client receives the m_{IPC2} messages inserted by the attacker before receiving those sent by the server. Here, the former always fails because the attacker obtains the message from the server over the IPC2 channel under our assumed attack model, as described in Section IV-C. Therefore, we consider only the latter. The following is a summary of the attacker's actions to avoid detection in each detection event.

(3-1): In Step 3 (Index exchange), the attacker repeatedly inserts index information, and the client receives m_{IPC2} index information from the attacker.

(3-2), (3-3): In Step 5 (Key verification), the attacker repeatedly inserts a MAC value, and the client receives m_{IPC2} MAC values from the attacker.

(3-4): In Step 6 (Attacker detection), the attacker repeatedly inserts the attacker detection signal to inform the client that there is no attacker, and the client receives m_{IPC2} attacker detection signals from the attacker.

3) ATTACK SCENARIOS IN EACH CASE

We summarize the specific attack scenarios in Sections VI-B1 and VI-B2 below.

Attack Scenario in Case (a):

- In Step 2 (Random number exchange):

(a-1): The attacker "eavesdrops." The attacker obtains at least m_{IPC1} random number information sent by the server, and the client obtains m_{IPC1} random number information eavesdropped by the attacker.

(a-2): The attacker "eavesdrops." The attacker eavesdrops on at least m_{IPC1} of the random number information sent by the client, and the server obtains the m_{IPC1} random number information eavesdropped by the attacker.

- In Step 3 (Index exchange):

(a-3): The attacker "receives" index information sent by the server once.

(a-4): The attacker "receives" index information sent by the client once.

Attack Scenario in Case (b):

- In Step 2 (Random number exchange):

(b-1): The attacker "inserts" random number information. The client obtains m_{IPC1} random number information from the attacker.

(b-2): The attacker "eavesdrops." The attacker eavesdrops on at least m_{IPC1} of the random number information sent by the client, and the server obtains the m_{IPC1} random number information eavesdropped by the attacker.

- In Step 3 (Index exchange):

(b-3): The attacker "receives" index information sent by the server once.

(b-4): The attacker repeatedly "inserts" index information associated with the random number obtained from the client in Step 2, and the client obtains m_{IPC2} index information sent by the attacker.

- In Step 5 (Key verification):

(b-5): The attacker repeatedly inserts MAC value, and the client receives m_{IPC2} MAC values from the attacker.

- In Step 6 (Attacker detection):

(b-6): The attacker repeatedly inserts the attacker detection signal to inform the client that there is no attacker, and the client receives m_{IPC2} attacker detection signals from the attacker.

Attack Scenario in Case (c):

- In Step 2 (Random number exchange):

(c-1): The attacker "eavesdrops." The attacker obtains at least m_{IPC1} random number information sent by the server, and the client obtains m_{IPC1} random number information eavesdropped by the attacker.

(c-2): The attacker "receives" m_{IPC1} random number information sent by the client.

- In Step 3 (Index exchange):

(c-3): The attacker "receives" index information sent by the client once.

(c-4): It is the same as (b-4).

- In Step 5 (Key verification):

(c-5), (c-6): These are the same as (b-5) and (b-6), respectively.

Attack Scenario in Case (d):

- In Step 2 (Random number exchange):

(d-1): The attacker "inserts" random number information. The client obtains m_{IPC1} random number information from the attacker.

(d-2): It is the same as (c-2).

- In Step 3 (Index exchange):

(d-3), (d-4): These are the same as (c-3), (b-4), respectively.

- In Step 5 (Key verification):

(d-5), (d-6): These are the same as (b-5), (b-6), respectively.

C. ATTACK SUCCESS PROBABILITY

This section explains the attack success probabilities of attack scenarios in Section VI-B3 based on the communication channel model defined in Section IV.

Remind that $P_{M(e)}[A]$ and $P[M(e)]$ are probabilities that the receiver Bob receives a message from the sender Alice and the attacker Mallory when Malory eavesdrops, respectively. Similarly, $P_{M(s)}[A]$ and $P[M(s)]$ are probabilities that the receiver Bob receives a message from the sender Alice and the attacker Mallory when Malory inserts messages, respectively. We denote the attack success probabilities in Cases (a) - (d) using these probabilities in the following. Note that we neither use $P_{M(e)}[A]$ nor $P_{M(s)}$ in practice because $P_{M(e)}[A] + P[M(e)] = 1$ and $P_{M(s)}[A] + P[M(s)] = 1$ hold.

Attack Scenario in Case (a):

The success probabilities of each event are given as follows:

(a-1): $P[M(e)]^{m_{IPC1}}$
(a-2): $P[M(e)]^{m_{IPC1}}$
(a-3): 1
(a-4): 1

Therefore, the attack success probability is

$$P_{Case\ (a)} = P[M(e)]^{2m_{IPC1}}.$$

Attack Scenario in Case (b):

The success probabilities of each event are given as follows:

(b-1): $P[M(s)]^{m_{IPC1}}$
(b-2): $P[M(e)]^{m_{IPC1}}$
(b-3): 1
(b-4): $P[M(s)]^{m_{IPC2}}$
(b-5): $P[M(s)]^{m_{IPC2}}$
(b-6): $P[M(s)]^{m_{IPC2}}$

Therefore, the attack success probability is

$$P_{Case\ (b)} = P[M(e)]^{m_{IPC1}} \cdot P[M(s)]^{m_{IPC1}} \cdot P[M(s)]^{3m_{IPC2}}.$$

Attack Scenario in Case (c):

The success probabilities of each event are given as follows:

(c-1): $P[M(e)]^{m_{IPC1}}$
(c-2): 1
(c-3): 1
(c-4) - (c-6): These are the same as (b-4) - (b-6), respectively.

Therefore, the attack success probability is

$$P_{Case\ (c)} = P[M(e)]^{m_{IPC1}} \cdot P[M(s)]^{3m_{IPC2}}.$$

Attack Scenario in Case (d):

The success probabilities of each event are given as follows:

(d-1): $P[M(s)]^{m_{IPC1}}$
(d-2) - (d-6): These are the same as (c-2), (c-3), (b-4) - (b-6), respectively.

Therefore, the attack success probability is

$$P_{Case\ (d)} = P[M(s)]^{m_{IPC1}} \cdot P[M(s)]^{3m_{IPC2}}.$$

These attack success probabilities can be reduced by increasing m_{IPC1} and m_{IPC2}, i.e., it indicates that KA-IPC has a trade-off between execution time, which increases proportionally to the number of exchanging messages, and security.

VII. EVALUATION OF KA-IPC IMPLEMENTATION

As described in Sections IV and VI, this paper evaluates the security of KA-IPC in terms of the attack success probability. KA-IPC can set the numbers of sending and receiving as security parameters, respectively. The attack success probabilities are calculated based on the behaviors of the communication channels using KA-IPC and the security parameters. Since the communication channels used in KA-IPC are based on the characteristics of an actual OS, the behavior of the channel is likely to be highly dependent on the OS. In this paper, we experimentally examine the behavior on Linux and estimate the security parameters to perform KA-IPC securely, i.e., to satisfy 128-bit security.

A. EVALUATION ENVIRONMENT

As in Section IV-C2, we chose Raspberry pi 3 and Raspbian [31] as the experimental environment. Raspbian is one of Linux with a full-function like PC Linux, and Raspberry pi 3 and Raspbian are widely used in IoT devices. By experimentally verifying that the overhead of KA-IPC is small on the environment, we demonstrate that the applicable scope of this technology is larger than such a scope of [13], which includes high-end devices such as servers and multi-user PCs.

We run three processes: a sender that repeatedly sends messages, a receiver that repeatedly receives messages, and an attacker that repeatedly eavesdrops or inserts messages. The sender and receiver run with a non-privileged user in the same user session. This is based on the assumption that an actual application using IPC, such as the password manager using IPC, denoted in Section I. The attacker runs with a non-privilege user in a different user session as the sender and receiver.

B. PARAMETERS FOR EXPERIMENT

In KA-IPC, the process priority and the frequency of exchanging messages intuitively seem to influence the attack success probabilities. In Raspbian and other Linux OSs, the process priority can be determined by a nice value set by nice command [33], and the frequency can be set by inserting sleep function [34]. We experimentally examined that whether the priority and frequency affect the attack success probabilities. As a result, we confirmed that the process priority does not essentially affect the attack success probabilities, but whether the insertion of the sleep function does. When the sleep function is not inserted, the values of $P[M(e)]$ and $P[M(s)]$ are smaller, i.e., the number of messages sending and receiving messages in each step can be reduced so that the execution time of KA-IPC can be shortened. For this reason, it is better not to insert the sleep function in KA-IPC. Appendix A shows the details of the experimental results. This section describes the experimental results when the following parameters are used, and then we estimate the security parameters to satisfy 128-bit security under the execution environment using the result.

Process priority:

The sender, receiver, and attacker are 0.

Frequency of exchanging messages:

The sender, receiver, and attacker do not insert sleep function.

C. EXPERIMENTAL DETAILS

The attack success probabilities in Section VI can be calculated using the security parameters (the numbers of receiving m_{IPC1} and m_{IPC2}) and the probabilities $P[M(e)]$ and $P[M(s)]$. These probabilities are dependent on the behavior of the communication channels. In this section, we experimentally obtain these probabilities.

KA-IPC must be decided both the number of receiving and sending messages as a protocol specification. The numbers of sending n_{IPC1} and n_{IPC2} over each communication channel IPC1 and IPC2 can be calculated from the numbers of receiving m_{IPC1} and m_{IPC2} and the number of sending messages during t, $n(t)$, as follows: $n_{IPC1} = n(t) \cdot m_{IPC1}$ and $n_{IPC2} = n(t) \cdot m_{IPC2}$. Since $n(t)$ is dependent on the communication channel, we experimentally obtain $n(t)$ in this section. We explain the details of experiments. We finished the measurement in each experiment when the receiver receives 1,000 messages, and we measured 100 trials.

Experiment 1 (Measurement of $P[M(e)]$): The attacker repeatedly eavesdropped. The attacker created the same UDP socket as the receiver and eavesdropped on the UDP socket channel. The attacker and the receiver each recorded the received messages. By comparing each received message, we calculated the probability that the receiver's messages contain attacker's messages contains, as $P[M(e)]$.

Experiment 2 (Measurement of $P[M(s)]$): The attacker repeatedly inserted messages to the receiver. The receiver recorded the received messages. We calculated the probability that the receiver's messages contain the messages sent by the attacker, as $P[M(s)]$.

Experiment 3 (Measurement of $n(t)$): The attacker repeatedly inserted a message to the receiver. The sender sent a value as a message to the receiver while counting the value. The receiver recorded the received messages. We calculated the total number of messages sent by the sender from the received messages, and then calculated the number of sent messages until the receiver receives one message, as $n(t)$.

D. EXPERIMENTAL RESULTS

Table 3 and Figure 8 show the results of each experiment. In Section VII-E, we estimate the security parameters of KA-IPC using the worst values of $P[A(e)]$ and $P[A(s)]$

TABLE 3. Probabilities obtained from experiments 1-3 in UDP socket.

Experiment	Average	Worst-case
1 ($P[M(e)]$)	0.0099	0.021
2 ($P[M(s)]$)	0.45	0.51
3 ($n(t)$)	2.70	3.88

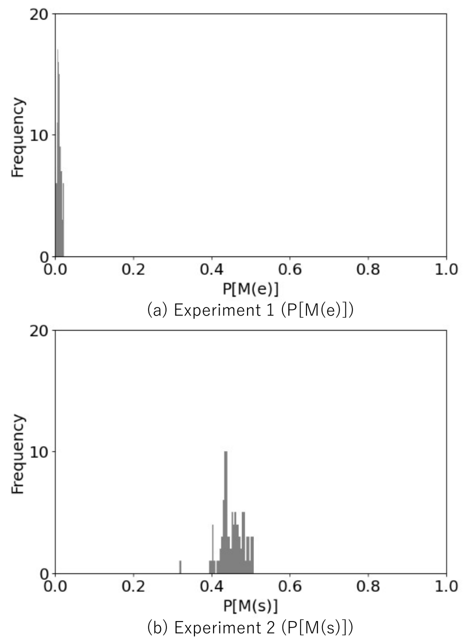


FIGURE 8. Histogram of experimental results.

obtained from each experiment and the smallest integer above the worst value of $n(t)$, $n(t) = \lceil 3.88 \rceil = 4$.

E. SECURITY PARAMETERS

The security of KA-IPC depends on the behaviors of the communication channels and the security parameters, m_{IPC1} and m_{IPC2}, which are the numbers of receiving over the IPC1 channel and IPC2 channel. We estimate the appropriate security parameters using the experimental results in Section VII-D. This section considers the security parameters for KA-IPC to achieve 128-bit security, as described in Section IV. For simple discussion, we assume that the random numbers are 128 bit in Step 2 and that the attacker fails the attack if the attacker does not know all the random numbers used for the client's key generation. For KA-IPC to achieve 128-bit security, the attack success probability shown in Section VI must satisfy the following inequality.

- $P_{Case\ (a)} \leq 2^{-128}$
- $P_{Case\ (b)} \leq 2^{-128}$
- $P_{Case\ (c)} \leq 2^{-128}$
- $P_{Case\ (d)} \leq 2^{-128}$

When we focus on the exponential parts of the above equations, they are simultaneous linear inequalities. We consider the way to select security parameters that satisfy the above inequalities while reducing the execution time of the protocol. Since the bottleneck in the execution time is repeatedly exchanging messages, the approximate execution time can be estimated as $2m_{IPC1} + 6m_{IPC2}$. Therefore, selecting the optimal security parameters can be regarded as a linear programming problem to find the minimum value of $2m_{IPC1}+6m_{IPC2}$ (and m_{IPC1} and m_{IPC2} that gives it) after satisfying the above inequalities. We calculated m_{IPC1} and m_{IPC2} using the experimental values in Section VII-D. As a result, m_{IPC1} is 12, m_{IPC2} is 40, n_{IPC1} is 48 and n_{IPC2} is 160. The execution time for $2m_{IPC1} + 6m_{IPC2}$ in this experimental environment was about 8.5 ms. From the results, we confirmed that KA-IPC could be implemented in a sufficiently realistic time.

VIII. CONCLUSION

This paper showed that it is possible to construct an uncertain communication channel using UDP sockets as IPC and proposed a key agreement protocol KA-IPC using this channel. We also experimented on a Raspberry pi, which is not a rich environment, and showed that KA-IPC could realize secure key sharing within 8.5 ms. This result suggests that KA-IPC can be used not only in rich environments but also in mid-range devices such as IoT devices and smartphones. KA-IPC is expected to be useful for applications that work in conjunction with a web browser, such as password managers, where real-time is not essential. Although the way to share a key for cryptographic communication between processes has not been established, KA-IPC has shown it. Using TCP sockets (or UDP sockets) for cryptographic communication between processes using a cryptographic key shared using KA-IPC, it is possible to protect communication messages from the attacker who interrupts the protocol and impersonates the server, such as the socket hijacking attack.

The basic concept of KA-IPC is to utilize the uncertainty of the communication channel caused by a general OS's characteristics for the key agreement. Although this paper describes the implementation of KA-IPC using UDP sockets, it can be applied to any IPC method. KA-IPC may be effective in situations where access control mechanisms cannot be used for IPC protection. The specific implementation of KA-IPC under other IPC mechanisms and attack models and its evaluation are future works.

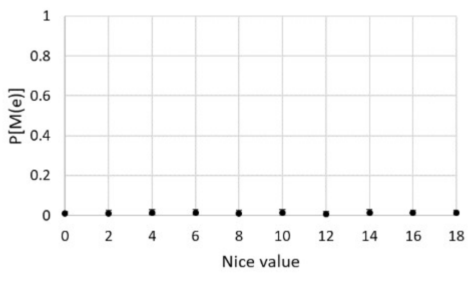
This paper suggests the possibility of using various cryptographic techniques in IPC rather than simply enabling cryptographic communication. It is necessary to adjust the security parameters for each OS to use KA-IPC in practice. Future work includes evaluation under more diverse environments to clarify the recommended parameters.

APPENDIX A
PARAMETERS FOR EXPERIMENTS

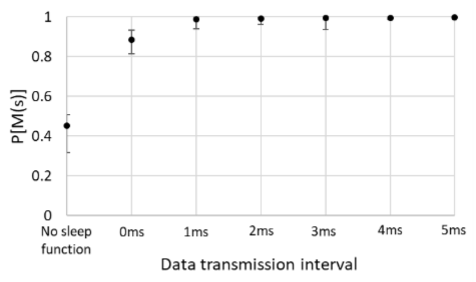
This section shows the results of some preliminary experiments to investigate an uncertain channel with UDP sockets. We experimentally investigate the influence of the parameters described in Section VII, which are process priority and frequency of exchanging messages, and we reveal whether these parameters affect the security of KA-IPC. From the experiments, we found that only the frequency of exchanging messages affects the security of KA-IPC. In the following, we describe the outline and results of each experiment.

TABLE 4. Attackers and countermeasures.

Application type	Application version	Browser, extension version	Communication channel	Attack	Access control	KA-IPC
Pasword managers	RoboForm 8.4.4	Chrome, 8.4.3.6 Firefox, 8.4.3.4 Safari, 8.4.5	Network socket	Client impersonation		
	Dashlane 5.1.0	Chrome, 5.5.3 Firefox, 5.5.3 Safari, 5.5	Network socket	Server impersonation		✓
	1Password 6.8.4	Safari, 4.6.12	Network socket	Server impersonation		✓
	F-Secure Key 4.7.114	Chrome, 1.0.0.3 Firefox, 1.0.3	Network socket Network socket	Server impersonation Client impersonation		✓
	Password Boss 3.1.34	Chrome, 3.1.3434 Firefox, 3.31.3434	Named pipe	Man in the middle	✓	✓
	Sticky Password 8.0.4	Chrome, 8.0.12.120 Firefox, 8.0.12.130 Safari, 8.0.2.63	Network socket	Server impersonation Client impersonation		✓
Backends with	Bizzard 1.10.1.9799		Network socket	Client impersonation		
HTTP API	Transmission 2.93		Network socket	Client impersonation		
	Spotify 1.0.73.345		Network socket	Client impersonation		
Others	My SQL		Named pipe	Man in the Middle	✓	✓
	Keybase 1.0.40		Named pipe	Server impersonation Client impersonation		✓

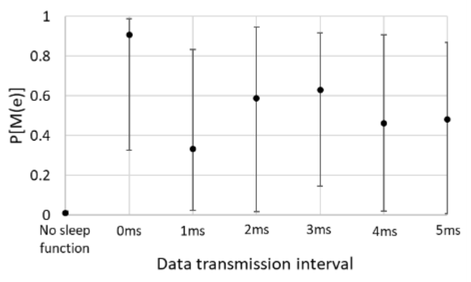


(a) Experiment 1 (P[M(e)])

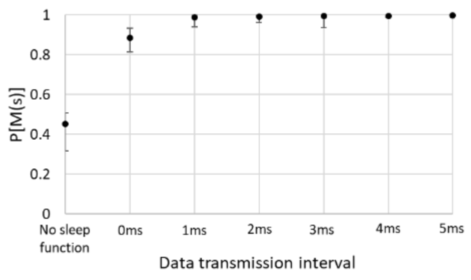


(b) Experiment 2 (P[M(s)])

FIGURE 9. Effect of process priority on communication channel.



(a) Experiment 1 (P[M(e)])



(b) Experiment 2 (P[M(s)])

FIGURE 10. Effect of frequency of exchanging messages on communication channel.

A. PROCESS PRIORITY

The process priority is one of the parameters that an attacker can set, and may affect the security of KA-IPC by affecting the execution order of the processes. Although a non-privileged user does not control most of the parameters that affect the execution order of processes, the user can set a process priority as a nice value of process using the nice command in Linux OSs. The range of nice value is from -19 to $+19$, and the smaller value indicates the higher process priority. The default nice value is 0. A non-privileged user can set the nice value of processes, which run in their user session, from 0 to $+19$, i.e., lower the process priority.

We experimentally investigated the effect of the process priority on the behavior of the uncertain communication channel with a UDP socket. We performed Experiments 1 and 2 in Section VII under conditions where the attacker varied his process priority from 0 to $+19$, and the sender and receiver are default nice value 0. Figure 9 shows the experimental results. From Fig. 9, we confirmed that the process priority does not affect the behavior of the uncertain communication channel. Therefore, there is no need to consider the process priority for the security of KA-IPC under the environment used in this experiment.

B. FREQUENCY OF EXCHANGING MESSAGES

In KA-IPC, the client and server can set the interval between sending and receiving messages by inserting the sleep function. We experimentally investigate the effect of the frequency of exchanging messages on the uncertain communication channel. We inserted the sleep function for each sending and receiving a message by the sender and receiver, respectively, and performed Experiments 1 and 2 in Section VII while varying the sleep time from 0 ms to 5 ms in 1 ms increments. The sleep function was not inserted for the attacker at this time. We also experimented with the case where the sender and receiver did not insert the sleep function. Figure 10 shows the results of the experiments. From Fig. 10, we confirmed whether inserting sleep function affects the communication channel; without the insertion of the sleep function, $P[A(e)]$ and $P[A(s)]$ are smaller. In other words, the values of the security parameters in Section VII can be made smaller to satisfy 128-bit security, and thus the execution time of KA-IPC can be shortened. Therefore, it is better not to insert the sleep function under the environment used in the experimental environment.

APPENDIX B
USEFULNESS OF KA-IPC

In this section, we describe the usefulness of KA-IPC for real applications. Table 4 shows the applications have vulnerable to impersonating attacks in [13]. While the conventional countermeasure, access control, are effective in few cases, proposed KA-IPC is effective in many cases.

REFERENCES

[1] W. R. Stevens, *UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI*. Upper Saddle River, NJ, USA: Prentice-Hall, 1998.

[2] W. R. Stevens, *UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications*. Upper Saddle River, NJ, USA: Prentice-Hall, 1999.

[3] *Named Shared Memory*. [Online]. Available: https://docs.microsoft.com/ja-jp/windows/win32/memory/creating-named-shared-memory?redirectedfrom=MSDN

[4] *Named Pipes*. [Online]. Available: https://docs.microsoft.com/ja-jp/windows/win32/ipc/named-pipes

[5] *Message Queues*. [Online]. Available: https://docs.microsoft.com/ja-jp/windows/win32/winmsg/about-messages-and-message-queues?redirectedfrom=MSDN

[6] *McAfee Labs COVID-19 Threats Report*. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-july-2020.pdf

[7] Y. Shao, J. Ott, Y. J. Jia, Z. Qian, and Z. M. Mao, "The misuse of Android unix domain sockets and security implications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 80–91.

[8] L. Xing, X. Bai, T. Li, X. Wang, K. Chen, X. Liao, S.-M. Hu, and X. Han, "Cracking app isolation on apple: Unauthorized cross-app resource access on MAC OS X and iOS," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 31–43.

[9] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proc. 9th Int. Conf. Mobile Syst., Appl., Services*, 2011, pp. 239–252.

[10] G. Cohen, "Call the plumber you have a leak in your (named) pipe," in *Proc. CON*, 2017, pp. 1–50.

[11] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "CHEX: Statically vetting Android apps for component hijacking vulnerabilities," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 229–240.

[12] B. Watts. *Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit*. Accessed: Dec. 9, 2019.

[13] T. Bui, S. P. Rao, M. Antikainen, V. M. Bojan, and T. Aura, "Man-in-the-machine: Exploiting ill-secured communication inside the computer," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1511–1525.

[14] *SE Linux*. [Online]. Available: https://www.nsa.gov/what-we-do/research/selinux/

[15] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948.

[16] A. D. Wyner, "The wire-tap channel," *Bell Syst. Tech. J.*, vol. 54, no. 8, pp. 1355–1387, 1975.

[17] U. M. Maurer, "Secret key agreement by public discussion from common information," *IEEE Trans. Inf. Theory*, vol. 39, no. 3, pp. 733–742, May 1993.

[18] C. H. Bennett, G. Brassard, and J.-M. Robert, "Privacy amplification by public discussion," *SIAM J. Comput.*, vol. 17, no. 2, pp. 210–229, Apr. 1988.

[19] G. Brassard and L. Salvail, "Secret-key reconciliation by public discussion," in *Proc. Workshop Theory Appl. Cryptograph. Techn. Adv. Cryptol.* Berlin, Germany: Springer-Verlag, 1994, pp. 410–423.

[20] C. H. Bennett, G. Brassard, C. Crépeau, and U. M. Maurer, "Generalized privacy amplification," *IEEE Trans. Inf. Theory*, vol. 41, no. 6, pp. 1915–1923, Nov. 1995.

[21] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," *Theor. Comput. Sci.*, vol. 560, pp. 7–11, 2014.

[22] A. K. Ekert, "Quantum cryptography based on Bell's theorem," *Phys. Rev. Lett.*, vol. 67, no. 6, pp. 661–663, Aug. 1991.

[23] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, "Quantum cryptography," *Rev. Modern Phys.*, vol. 74, no. 1, pp. 145–195, Mar. 2002.

[24] V. Aggarwal, L. Lai, A. R. Calderbank, and H. V. Poor, "Wiretap channel type ii with an active eavesdropper," in *Proc. IEEE Int. Symp. Inf. Theory*, Feb. 2009, pp. 1944–1948.

[25] P. Wang and R. Safavi-Naini, "A model for adversarial wiretap channels," *IEEE Trans. Inf. Theory*, vol. 62, no. 2, pp. 970–983, Feb. 2016.

[26] P. Wang, R. Safavi-Naini, and F. Lin, "Erasure adversarial wiretap channels," in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2015, pp. 1061–1068.

[27] R. Liu and W. Trappe, *Securing Wireless Communication at Physics Layer*. Springer, 2009.

[28] A. A. Hassan, W. E. Stark, J. E. Hershey, and S. Chennakeshu, "Cryptographic key agreement for mobile radio," *Digit. Signal Process.*, vol. 6, no. 4, pp. 207–212, 1996.

[29] T. Aono, K. Higuchi, T. Ohira, B. Komiyama, and H. Sasaoka, "Wireless secret key generation exploiting reactance-domain scalar response of multipath fading channels," *IEEE Trans. Antennas Propag.*, vol. 53, no. 11, pp. 3776–3784, Nov. 2005.

[30] A. Niakanlahiji, J. Wei, MR. Alam, Q. Wang, and B. T. Chu, "Shadowmove: A stealthy lateral movement strategy," in *Proc. 29th Secur. Symp.*, 2020, pp. 559–576.

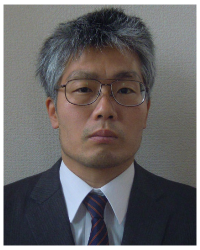
[31] *Raspberry*. [Online]. Available: https://www.raspberrypi.org/

[32] H. Krawczyk, "Cryptographic extraction and key derivation: The HKDF scheme," in *Advances in Cryptology*, T. Rabin, Ed. Berlin, Germany: Springer, 2010, pp. 631–648.

[33] *Nice Command*. [Online]. Available: https://man7.org/linux/man-pages/man1/nice.1p.html

[34] *Usleep Command*. [Online]. Available: https://man7.org/linux/man-pages/man3/usleep.3.html

MANAMI SUZUKI received the B.E. degree in information engineering and the M.S. degree in information sciences from Tohoku University, Sendai, Japan, in 2016 and 2018, respectively. She is currently with Yokohama Research Laboratory, Hitachi Ltd. Her research interests include hardware security and physically unclonable functions.



DAI WATANABE received the B.S. and M.S. degrees from Tohoku University, Sendai, Japan, in 1994 and 1996 respectively, and the Ph.D. degree from Tokyo University of Science, in 2007. Since 1999, he has been engaged in research on information security, cryptography and cryptographic protocol with the Research and Development Group, Hitachi Ltd. He is a member of the Information Processing Society of Japan (IPSJ) and The Institute of Electronics, Information and Communication Engineers (IEICE).

TSUTOMU MATSUMOTO (Member, IEEE) received the Doctor of Engineering degree from The University of Tokyo, in 1986. He is currently a Professor with the Faculty of Environment and Information Sciences, Yokohama National University, and the Director of the Cyber Physical Security Research Center, National Institute of Advanced Industrial Science and Technology. He has been interested in research and education of embedded security systems, such as the IoT devices, cryptographic hardware, in-vehicle networks, instrumentation and control security, tamper resistance, biometrics, artifact-metrics, and countermeasure against cyber-physical attacks. He received the IEICE Achievement Award, the DoCoMo Mobile Science Award, the Culture of Information Security Award, the MEXT Prize for Science and Technology, and the Fuji Sankei Business Eye Award. He serves as the Chair for the Japanese National Body for ISO/TC68 (Financial Services) and the Cryptography Research and Evaluation Committees (CRYPTREC) and as an Associate Member for the Science Council of Japan (SCJ).

NAOKI YOSHIDA received the M.S. and Ph.D. degrees in informatics from Yokohama National University, in 2014 and 2017, respectively. He is currently a Specially Appointed Assistant Professor with the Institute of Advanced Sciences, Yokohama National University. His research interests include embedded system security, artifact metrics, and instrumentation security.

JUNICHI SAKAMOTO received the M.I.S. and Doctor of Informatics degrees from Yokohama National University, Japan, in 2017 and 2020, respectively. He is currently working as a Postdoctoral Researcher with Yokohama National University. He has engaged in various researches regarding information security, including the methodology of secure implementation of the cryptographic algorithms, side-channel attacks to pairing computation, and laser-based fault attacks.