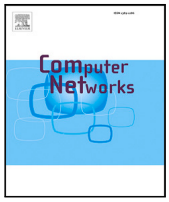
Computer Networks 193 (2021) 108055

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



A trusted and collaborative framework for deep learning in IoT

Qingyang Zhang [a,b], Hong Zhong [a,b,*], Weisong Shi [c], Lu Liu [d]

[a] School of Computer Science and Technology, Anhui University, China
[b] Anhui Engineering Laboratory of IoT Security Technologies, Anhui University, China
[c] Department of Computer Science, Wayne State University, USA
[d] Department of Informatics, University of Leicester, UK

ARTICLE INFO

Keywords:
Collaborative framework
Trusted execution environment
Deep learning
Internet of Things

ABSTRACT

More and more Internet of Things (IoT) applications provide intelligent services, with the development of artificial intelligence algorithms, such as deep reinforcement learning. However, along with the trend of utilizing a large model with high accuracy in AI-enabled IoT, resource-limited IoT devices are difficult to handle these large-scale models with high response latency. By collaborating with edge nodes, the devices could respond quickly. However, IoT applications contain a large amount of user privacy, and pushing data to others might lead to privacy leakage. Inspired by the trusted execution environment technology, we propose a framework that enables trusted collaboration for future AI-enabled IoTs, in terms of computation security and transmission security, where the data could be processed in an isolated environment, and two approaches are proposed to ensure the security in data transmission. Experimental results show that our framework provides flexible and dynamic collaboration with low overhead and can effectively support collaborative edge intelligence.

1. Introduction

With the growth of the Internet of Things (IoT), more and more IoT devices are being deployed around people and enrich our daily life [1,2]. According to the report from Ericsson [3], the IoT connections will reach 26.9 billion by 2026. The IoT systems could be intelligent by analyzing sensed data using artificial intelligence (AI) algorithms, such as deep learning [4,5], deep reinforcement learning (DRL) [6], etc. For example, the Amazon Alexa could understand people's speech and make a response, which acts as an intelligent assistant in a smart home system. In traditional IoT systems, sensed data is often uploaded to the cloud that is analyzed using AI algorithms. However, the data transmission causes a high response time [7]. With the development of AI technology, especially the introduction of DRL [6], things would be smarter. For instance, autonomous vehicles rely on powerful AI algorithms to analyze the surrounding environment [8,9], and DRL is one of the solutions, which provides an end-to-end way to train models [10]. But, the complexity of the environment also makes the model larger. In this case, with the requirement of ultra-low latency, autonomous vehicles have to be equipped with high-performance hardware. However, it is difficult to infer such large models on resource-constrained IoT devices.

Typically, these resource-constrained IoT devices have to update their sensed data to the cloud for analysis with high latency. And with billions of devices, the data is huge, resulting in a significant burden on cloud centers. Edge computing [7,11,12], as a new computing paradigm, is promising to reduce the burden of the cloud and the response time of one service via the collaboration of end devices, edge nodes, and the cloud. As shown in Fig. 1, the end devices could offload the computation task to the nearby edge nodes with a lower transmission latency, resulting in a slower response time [13–15]. Taking the killer application, edge video analytics, as an example, city cameras could send areas to the cloud for further analysis, detected by a small AI model, thus avoid the high latency and energy consumption associated with running a large-scale AI model by them self [16,17]. However, IoT data typically contains a significant amount of user privacy, and user privacy could be learned from these data [18–20]. Edge nodes typically do not have the same level of security protection as cloud centers, and processing the data on such edge nodes might lead to a high risk of privacy leakage. Thus, how to guarantee the security of data and protect user privacy in a collaborative edge-cloud environment in future AI-enabled IoT is an issue that must be addressed.

Homomorphic encryption technology [21] could process data in encrypted without any knowledge about data so that it is usually used for outsourcing service in the cloud. Thus, in edge collaboration environment, the IoT devices could outsource data to the edge or cloud with homomorphic encryption technology. However, the performance is a big issue [22], especially for data encryption/decryption

* Corresponding author at: School of Computer Science and Technology, Anhui University, Hefei 230601, China.
E-mail address: zhongh@ahu.edu.cn (H. Zhong).

https://doi.org/10.1016/j.comnet.2021.108055
Received 20 November 2020; Received in revised form 3 March 2021; Accepted 23 March 2021
Available online 31 March 2021

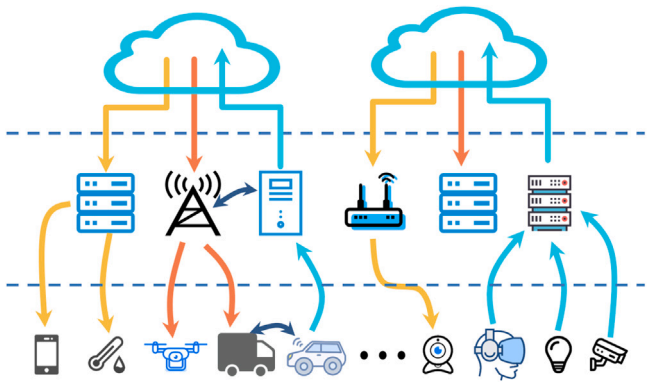


Fig. 1. Collaborative edge intelligence in AI-enabled IoT.

on the resource-constrained IoT devices and large AI model inferring and training [23] on the edge. Recently, hardware-assisted trusted execution environment (TEE), such as Intel Software Guard eXtensions (SGX), has become increasingly popular [24,25], which could provide a trusted, isolated, and secure execution environment for application thus process data thus ensure the security of data in computing. Many TEE-enhanced frameworks [24,26] are proposed. However, most of them aim to provide a runtime or programming interface for application in the local machine without any collaborative computing support, such as Asylo [27], Occlum [28]. Part of them could provide a collaborative mechanism between two parties. However, currently, no one framework could provide a TEE-enabled collaboration in edge-enabled IoT.

In this paper, we aim to design a trusted and collaborative framework for future AI-enabled IoT, called *TCFDL*. We have explained our motivating application and summarized its challenges and our Contributions in the following sections.

1.1. Motivating application

As a typical application of the intelligent IoT and edge computing, connected and autonomous vehicles (CAVs) [8,29,30] are equipped with various sensors, such as cameras, radar, LiDAR, global navigation satellite system, and analyze sensed data using computer vision and AI algorithms. To process data in time, CAVs have to be equipped with powerful computing devices worth tens of thousands of dollars [8]. As envisioned by [8,29], the CAVs will be a computer on the wheel that could install many third-party applications to improve ride experience. In this case, the computation burden will significantly increase. Thus, the collaboration between CAVs, as well as CAVs and edge nodes (such as roadside units) could reduce CAV's burden, by through sharing road information and recognition results of autonomous driving.

However, in-vehicle data and application data contain a large amount of user privacy, and transmitting this data to other edge nodes or CAVs for collaboration might lead to potential privacy leakage issues. Hence, if the security of data computing in a remote collaborative CAV or edge cannot be guaranteed, users may not be willing to perform collaborative computing.

1.2. Challenges

Although TEE could enable trusted and secure computing on the IoT devices, edges and cloud centers, it still have several barriers in a collaborative framework as followed.

Large-scale AI model in TEE. Typically, TEE can only provide limited resources for protected applications, such as memory. For example, Intel SGX only provides 90 MB of secure memory. In addition to normal data processing applications, the AI-related application must load the model file to memory then infer the model. In this case, the memory overhead will much larger than the size of protected memory in TEE, resulting in high latency overhead caused by frequent memory exchange between protected memory and normal memory. Thus, in the TEE-based trusted and collaborative framework, how to support large AI models with only low overhead is a challenge.

Transmission security between applications. Although data is protected when processed in TEE, the transmission between collaborative applications also might lack if no any protection mechanism. In our motivating application, CAVs have dynamic network topologies. That also leads to dynamic changes in the collaborative relationship. Typically, a scheduler could be used to dynamically schedule tasks between CAVs and edges. However, it leads to a challenge that keeping data confidentiality in the case of semi-trusted schedulers involved in message routing.

1.3. Our contribution

To address the mentioned barriers, we designed a trusted and collaborative framework for future AI-enabled IoT, which could ensure computation security and communication security in dynamic environments. Our contributions are as follows:

- We propose a framework with TEE support, which enables trusted and collaborative computing in edge-enabled IoT, especially for large-scale AI model training and inferring in future intelligent IoT applications and systems.
- We propose two different mechanisms to ensure data transmission security for dynamic computation offloading. We theoretically analyze the security of the proposed framework on computation and communication and evaluate the proposed framework's performance.
- A case study is used to show that the framework effectively improves the performance of large-scale deep learning in a collaborative environment.

The remainder of this paper is organized as follows. The designed trusted and collaborative framework is presented in Section 2, followed by data transmission security mechanisms in Section 3. Section 4 gives a short description of framework implementation. In Section 5, we evaluate the performances of the proposed framework, and a case study is performed in Section 6, followed by a discussion on security and scalability in Section 7. Finally, we review related works in Section 8 and conclude this paper in Section 9.

2. Architecture design

In this section, we will introduce the detailed design of the *TCFDL* framework, including terminologies, security & threat models, architecture and each components.

2.1. Terminology

We first introduce the terminologies that describe abstraction concepts in *TCFDL*.

- **Function**: Inspired by Function as a Service (FaaS), such as AWS Lambda, OpenFaaS, we also use a similar approach that all external services are implemented as functions in *TCFDL*, and accessed one service by its function name. For example, a video analytics application may externally provide a function that accepts video data and returns vehicle pictures in the video. Internally, the video analytics application relies on several functions, including video decoding and vehicle detection. As a result, the framework can automatically expand the application instance based on the number of video files to achieve parallel processing.

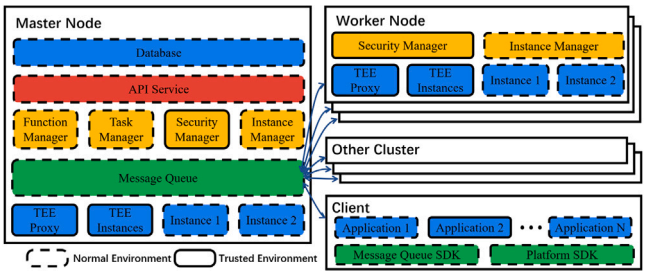


Fig. 2. The architecture of *TCFDL*.

- **Task**: In *TCFDL*, the request for one function will be a task, and its data is first transmitted to the framework, while the framework will offload the task data to a detail function instance. Also, when one application (function) collaborates with another one inside framework, it calls the collaborative function through the framework. Thus, the framework could schedule the task to an optimal one.

2.2. Security and threat model

First, we assume that the devices, edge nodes, and clouds are honest but curious. They do not interfere with the execution of applications and tamper with the data. But they are curious about the applications' data and always try to learn privacy information from the data. Correspondingly, applications do not fully trust the devices, edge nodes and clouds, but might trust the part of framework in TEE. In this case, TEE-based applications will always first validate the TEE part of the framework to ensure that the framework is trusted.

Based on the above assumptions, we set up two security models. In the first model, the application fully trusts the TEE and allows its data to be decrypted in the TEE part of the framework for task scheduling. In the second model, the application does not fully trust the TEE, and it does not want its data to be decrypted.

2.3. TCFDL Design

TCFDL is a trusted and collaborative framework for future AI-enabled IoT environments that enable TEE to preserve data on the platform in terms of computing and transmission. Fig. 2 present the architecture of *TCFDL*. The proposed framework includes three types of nodes, including Master Node (MNode), Worker Node (WNode), and client, as well as one centralized service as Service Manager (SMgr), which does not appear in the figure.

The master node manages all worker nodes in its cluster, and master nodes and worker nodes could host instances of applications implementing functions. All functions in a cluster should be registered to the Service Manager by the master node. In addition, one master node could serve as not only the master node in its cluster, but also a work node to its upper cluster, thus expose its inside functions to the public network through upper clusters, recursively. Moreover, the user client could query function-related providers through the Service Manager, then access functions based on *TCFDL* software development kit (SDK). Here, we first introduce different nodes and then introduce the components in these nodes.

2.3.1. Service manager

The Service Manager (SMgr) is usually deployed on a cloud to provide reliable services and act as a manager in the platform built by the proposed framework. Through this infrastructure, each cluster's functions are exposed for access by other clusters and client nodes. Thus, the SMgr accepts function registering queries, function unregistering, function inquiring from the master nodes of each cluster in the *TCFDL* platform, and clients. In addition, the SMgr acts as a trusted authority (TA) and is responsible for managing the identities, keys, and credentials of each node and client. Moreover, it also provides attestation service for TEE applications, including user applications and the TEE part of *TCFDL*.

2.3.2. Master node

The master node (MNode) is the manager of the cluster, including eight components, and is responsible for coordinating the worker nodes, application instances, and computational tasks within the cluster. The main functionalities of one master node are concluded as follows. First, it manages all functions and their instances inside the cluster, such as function provider registering and unregistering, and it will auto-scale the number of instances or migrate instances to an idle worker node. Second, all functions are event-driven, implemented based on an embedded message queue system. Thus, relationships between all functions within the local cluster and the underlying message queue topics are maintained by the master node, which also enables dynamic task offloading. Third, there exist two versions of the security management module in MNode, normal implementation without TEE support and TEE-enabled implementation. Thus, with hardware support, the master node starts a TEE proxy for TEE's special functionalities, such as authentication, message encryption for dynamic task offloading, etc. Finally, the cluster where the master node can also be used as a worker node within a parent cluster, exposing services to other networks in cases where it is not possible to communicate directly with other clusters (serving as a gateway).

2.3.3. Worker node

The Worker Node (WNode) mainly acts as a host node for instances managed by the master node. Therefore, the modules in a worker node are a subset in the master node, which contains only the security manager, instance manager, and TEE proxy. Here, the security manager module only includes the information used by itself, while the one in the master node store all information in the cluster. The instance manager module mainly accepts and responds to commands from the one in the master node. Besides, this module also reports to the master node on resource usage and status.

2.3.4. Client

By utilizing the provided software development kit, an application could be developed to access the services registered in the SMgr, where the application-hosted node is called client. Although our *TCFDL* is message-based, several traditional application-level protocols are provided, such as HyperText Transfer Protocol (HTTP), and the protocol will be translated in the receiving master node.

2.4. Components

As shown in Fig. 2, a node is structured by several components based on node type, *i.e.,* MNode, WNode, where MNode consists of eight components and worker node is with three components.

2.4.1. Function manager

The function manager module exists only in the master node. It is used to manage the functions in the cluster and keep them up-to-date with the SMgr. Suppose an instance of an application wants to provide a function in a cluster. In that case, it needs to register with the function manager module with related information, such as the name and configuration of the providing function. Once the function manager module receives the register information, it first determines the type of function, and retrieves whether the function with the same name exists or not. If the function does not exist, it will create the function with corresponding configurations, such as message routing information for the message queue module, and task scheduling related information for the task manager module. Then, it registers the function to the Service Manager. Thus other applications could access such function by its information. If the function exists, it only updates local information about the function.

2.4.2. Task manager

The task manager module monitors the status of each function's task in the message queue module, such as the number of tasks, the response time, and the number of pending tasks, and obtains the running status of function-related instances from the instance manager module. Firstly, based on collected information, the task manager module could dynamically schedule the function tasks according to the instances' status, thus implementing load balancing. Secondly, it will dynamically scale the function's instances. When the task response time or the number of pending tasks increases, it can be determined that the number of instances corresponding to the function cannot handle the task in time, and then it will send a command to the instance manager module to launch more instances. When a function has a large number of idle instances, it will reduce the number of instances. In addition, to support large-scale AI model inference in TEE, a TEE-enabled pure AI model inference application will be launched as multiple instances, and connected by a pipeline, which consists of several message queues created by the Task Manager module. Therefore, the size of model inferred by one TEE instance could be limited to avoid out of memory on TEE hardware.

2.4.3. Instance manager

The instance manager module is designed in both of the MNode and WNode, which has similar functionality and manages all instances on the node, such as creating, launching, restarting, or terminating instances. When a command is received from the task manager module in the master node, it will launch or terminate the program instance based on its configuration file. Simultaneously, the instance manager module collects all instance status, system status, and resource usage, such as CPU load, on the host node and reports them to the task manager node on the master node. So that the task manager module can choose the optimal worker node for the function instance scaling.

2.4.4. Security manager

The security manager module is designed in both of the MNode and WNode. When a worker node joins the cluster, the security manager module needs to verify the worker node's framework version, integrity, and other security parameters. Thus only legitimate, trustworthy, and secure worker nodes can join the cluster. Moreover, the security manager module needs to maintain secure channels between the nodes and verify the security of data and commands, as well as be responsible for the verification of function binary program files and related configuration files. Furthermore, it enables access control for the message queue module to protect data from unauthenticated access. In addition, in nodes that support TEE technology, the security manager module will run inside TEE, which can protect the security manager module from most attacks.

2.4.5. Message queue

In our framework, an embedded message queue system (EMQ) is used to transmit function data as well as status and commands between the master node and its worker nodes. A message queue system use queues for messaging, which allows applications to communicate by sending messages to each other. Typically, a topic is used to implement publish and subscribe pattern. Once a publisher publishes one message to the topic, corresponding subscribers will receive the message. Here, the function is similar to the concept of the topic. Thus, two collaborative applications could be connected by the function. It should be noted that the embedded message queue system provides communication channels not only inside one cluster but also between different clusters. That means one function instance could publish a message to another instance on a different cluster, and messages are automatically routed according to their function identities.

2.4.6. TEE proxy

The master node and worker node both contain a TEE proxy module, which is used to implement functionalities specific to the TEE technology. First, The integrity and validity of TEE-enabled instances need to be verified. These instances also can verify the security status of the hosted framework with the local TEE proxy. A proof will be provided so that these TEE-enabled instances could attest to each other, running in different clusters. Once the function is registered, the cluster will query security-related information from the SMgr (serving as TA here), such as keys for encrypting data if security communication is enabled. Later, during data transmission, the TEE proxy will assist the master node to process the encrypted data, such as decrypting or re-encrypting. The detailed secure data transmission approaches will be described in Section 3.

2.4.7. Database

The database module is implemented in the master node to store data within the cluster, such as information about functions, function program configuration files, cluster configuration, and framework logs. In the database module, multiple database systems and the file system are used. Structured data such as framework logs are stored in the traditional SQL database. Unstructured data (such as various configuration files) are stored in the file system. In contrast, these files' storage locations are stored in the SQL database. A key–value database system is used to reduce the access latency for these frequently-used data, such as routing information.

2.4.8. API service

In order to provide a friendly interface, *TCFDL* also introduces an API service module for users to configure the function and related programs. The API service module also provides APIs for the administrator to configure the cluster.

3. Security Data Transmission in *TCFDL*

Although TEE provides an isolated environment for function instances, which could protect task data from other applications in runtime. However, the data might be leaked or tampered with by a curious or malicious *TCFDL* node. Encryption and signature are used to deal with these problems, with the guarantees of data confidentiality and integrality. For example, the Transport Layer Security (TLS) protocol, which is currently used in popular Web architectures, obtains the security certificate of the web service after establishing the transmission channel, verifies the certificate, and then establishes the encryption key. Therefore, it requires that the data sender explicitly knows information such as the identity of the function instance. However, it is difficult in our scenarios with dynamic task offloading. Therefore, in order to solve the problem of secure data transmission, two approaches to secure data transmission are designed and implemented in the proposed framework. The first one is the hop-by-hop secure data transmission, which uses the traditional TLS approach enhanced by TEE, while the data needs to be decrypted and encrypted in TEE proxies. The second one is based on the proxy re-encryption technique, while the data do not be decrypted in transmission.

3.1. Hop-by-hop symmetrical encryption (HSE)

Fig. 3 illustrates the data transmission flows of our hop-by-hop symmetrical encryption approach. We assume that the users fully trust the component TEE proxy of *TCFDL*, thus they allow their data to be decrypted in TEE proxy. In the HSE approach, communicating TEE-enabled entities, including TEE proxies and TEE instances, will firstly establish symmetrical encryption keys via attestation operation. Then, the data is encrypted by data providing instance and published to the local EMQ module. Then the data should be decrypted and re-encrypted in all passing master nodes by their TEE proxies using the symmetrical

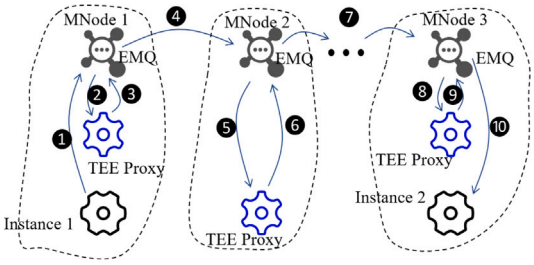


Fig. 3. Data transmission in hop-by-hop symmetrical encryption approach.

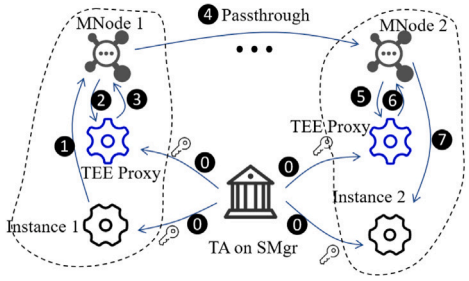


Fig. 4. Data transmission in end-to-end asymmetrical encryption approach.

encryption key with the next-hop TEE proxy (like ❷-❸, ❺-❻, ❽-❾, etc.). Finally, the encrypted data is transmitted to the data processing instance, and the later one could decrypt it using the encryption key with the local TEE proxy. In the above processes, all nodes except TEE proxies have no access to encrypted data.

Similar to Transport Layer Security (TLS), our HSE approach also utilizes Advanced Encryption Standard (AES) as the underlying symmetrical encryption algorithm. Here, a 128-bit key length version AES is used. It works under the GCM model, which could provide confidentiality and integrality for an encrypted message. In remained parts of this paper, we label it as AES-128.

3.2. End-to-end asymmetrical encryption (EAE)

Fig. 3 illustrates the data transmission flows of our end-to-end asymmetrical encryption approach. We assume that the users do not trust *TCFDL* including TEE proxy, thus they do not allow their data to be decrypted in TEE proxy. In this point, the proxy re-encryption technology is utilized in the EAE approach, in which a proxy (*i.e.,* TEE proxy in our *TCFDL*) is with the ability to transform a ciphertext encrypted under one public key into another ciphertext under another public key without leaking the underlying messages or private keys. As shown in Fig. 4, only TEE-enabled instances and their local TEE proxies process the data, and the data pass through in all passing master nodes. Here, an extra trusted authority is set for key management. In our *TCFDL*, the SMgr will act as the trusted authority. TEE-enabled instances should first query asymmetrical encryption keys from the TA on the SMgr. The TA will respond the key and send another key to its local TEE proxy for data re-encryption (❶). In our implementation, related application programming interfaces have been provided.

However, traditional asymmetrical encryption algorithms are with high computing overhead, comparing with symmetrical encryption algorithms when encrypting the same data. Thus, hybrid encryption is used here. The function data is encrypted by AES-128 (symmetrical encryption), and the AES-128 encryption key is encrypted by asymmetrical encryption. The used cryptographic operations are as follows.

Setup: By randomly choosing a k-bit prime q ($k = 160$ in our implementation), an additive cyclic group G compose of elliptic curve, where a generator P is set for group G. Thus, the public parameters consist of $\{k, q, P, G\}$. This algorithm is as a part work of ❶, and should be performed at the beginning of system start.

KenGen: This operation typically includes three sub-operations triggered by three case: (1) After Setup, the TA will choose a random value as function encryption key $SK_f = f$, where $f \in Z_q^*$; (2) When one instance i registers as data provider, the TA will generates a pair of keys, including private key $SK_i = x_i$ and public key $PK_i = x_i P$, where $x_i \in Z_q^*$, Then, the TA calculates re-encryption key $RK_{i\to f} = \frac{SK_f}{SK_i} = \frac{f}{x_i}$. Finally, the TA will respond to the instance and its local TEE proxy with $\{k, q, P, G, SK_i, PK_i\}$ and $\{k, q, P, G, RK_{i\to f}\}$, respectively, as shown in ❶; and (3) When one instance j registers as data processor, the TA will generate a pair of keys, SK_j and PK_j as before. Then, the TA calculates re-encryption key $RK_{f\to f}$, and sends these keys to the instance and its local TEE proxy along with system parameters.

Enc: This algorithm is used by data providing instances to encrypt function data. When one instance i sends one function data M, it must encrypt the data using symmetrical encryption algorithm (*i.e.,* AES-128), where the encryption key is Key. Then, the encryption key is encrypted as two parts $\{EK_{i,1}, EK_{i,2}\}$:

$$EK_{i,1} = (Key \| Padding) \oplus rP \tag{1}$$

$$EK_{i,2} = rSK_i P = rx_i P \tag{2}$$

where $Padding$ is a random binary string to pad Key to the same length with rP, and r is a random value. Finally, the instance could publish function data with new form of $\{EK_{i,1}, EK_{i,2}, AESEnc_{Key}(M)\}$ in the process ❷.

ReEnc: When a TEE proxy receives a function data encrypted by Enc or forwards data to a local TEE instance, it should use ReEnc to transform encrypted encryption key under one key to another key, using the key of $RK_{i\to f}$ in ❸-❹ or $RK_{f\to j}$ in ❻-❼. Here we take the calculation of re-encrypting using $RK_{i\to j}$ as an example.

$$EK_{j,1} = EK_{i,1} = (Key \| Padding) \oplus rP \tag{3}$$

$$EK_{j,2} = EK_{i,2} RK_{i\to j} = rx_j P \tag{4}$$

Dec: This algorithm is used by TEE instances to decrypt function data. When a TEE instance j receives data of $\{EKj, 1, EK_{j,2}, AESEnc(M)\}$ (in ❽), it first figures out symmetrical encryption key by calculating as follows:

$$Key \| Padding = EK_{j,i} \oplus (EK_{j,2} SK_j^{-1}) \tag{5}$$

Then, it could obtain plaintext function data from $AESEnc_{Key}(M)$ using encryption key Key.

4. Prototype implementation

In this section, we will introduce the implementation of a prototype based on our design. We implement a prototype of *TCFDL* using Golang language and C/C++ language, with a set of open sources, *e.g.,* NATS [31], NATS Streaming [32], Redis [33] and FastHTTP [34], where C/C++ language is only used to implement TEE proxy component and TEE manager component. For TEE, we utilize Intel SGX to implement. The TEE-related components are implemented using Intel SGX SDK 2.7.

In our implementation, we mainly built an overlay on top of NATS Streaming and NATS to enable communication with the dynamic task offloading support. The inside client is implemented to manage all topics in *TCFDL*. It provides a topic for out-cluster function requester by aggregate related in-cluster topics. Here, two communication patterns are provided. The first one is a traditional publish/subscribe pattern built on top of NATS Streaming with a message acknowledging, thus reliable one-to-many communication is enabled. The second one is the request/reply pattern built on top of NATS without message acknowledging. The instance could publish a request to function, and only one function instance will receive and respond to this request. Thus, the later one also could be to utilize several traditional network protocol, such as HTTP.

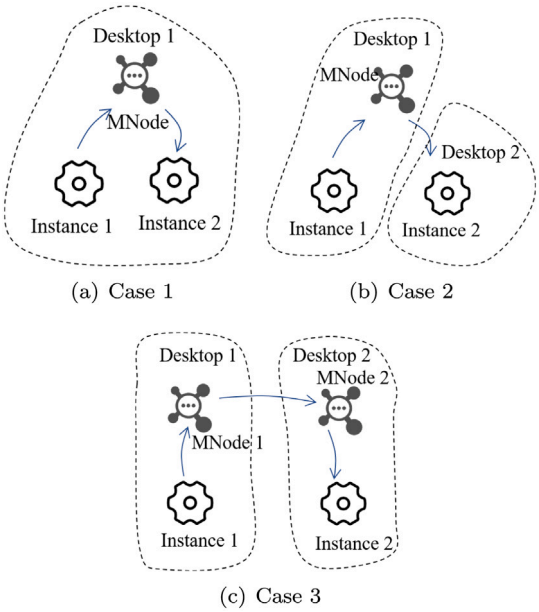


Fig. 5. Experimental topologies.

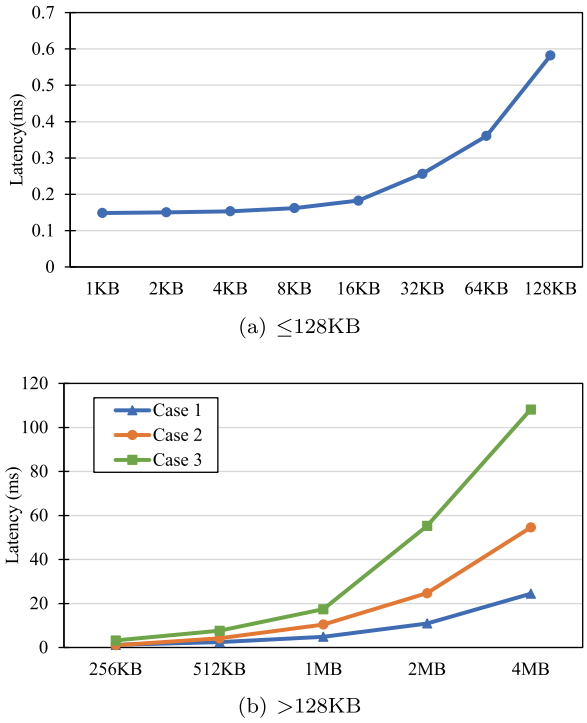


Fig. 6. Data transmission latency with different data size in difference cases.

5. Performance evaluation

In this section, we first introduce the experiment setup, followed by the performance evaluation. Then, a case study is performed to demonstrate collaboration in a dynamic CAV environment.

5.1. Experiment setup

To evaluate the performance of *TCFDL*, we set up three cases, as shown in Fig. 5. In the first case, two collaborative applications are hosted in the master of one cluster. In the second case, one of the applications is hosted in the master of one cluster, and another one is hosted in one worker node of the same cluster. In the last case, two applications are hosted in the master of two different clusters, respectively.

We set up a testbed with four desktops for these three cases, while all desktops are with an Intel Core i5-7400 running in performance mode at 3.0 GHz, and 4 GB memory. All the desktops are connected via the same switch, and the desktop as the Service Manager also provides time synchronization service. It should be noted that the data transmission latency for small volume is lower than the time error of two different desktops. Therefore, even though we have measured the transmission latency, it is not shown in the experimental results. All experiments are repeated 1000 times.

For secure data transmission, the symmetric cryptography used is Advanced Encryption Standard (AES) algorithm with a 128-bit security level and Galois/Counter Mode (GCM). The elliptic curve used in the experiments is the NIST Curve P-256, which also provides 128-bit security.

5.2. Communication

Fig. 6 shows data transmission performance for task offloading in three cases. Note that we just measure the data transmission latency here. It does not matter whether the data is encrypted or not. Due to the reason we mentioned in the experiment setup, we only present the performance of the Case 1 while the size of transmitted data is less than 128 KB. The results here are as expected. The latency of data transmission increases as the number of data increases, and Case 3 has the highest latency since the data has to be forwarded by two masters.

In our implementation, the difference between Case 1 and Case 2 is that task data have a cross-node communication, and the difference between Case 2 and Case 3 is that task data need to be forward two more times in local and remote collaborative master nodes. It means that the overhead caused by *TCFDL* is around 55 ms for the 4 MB data, and it only happens in cross-cluster collaborative. For the collaboration between applications in the same cluster, no overhead happens. In addition, it should be noted that cross-node communication in experiments actually does not consume much time, thus appearing to be a high additional overhead for our framework. Actually, cross-cluster communication latency is several times higher than the additional overhead of the framework. For example, as the study in [35], transmitting 4 MB data causes around 2.3 s and 4.4 s, respectively, for cloud servers 200 miles and 450 miles away.

5.3. Secure communication for dynamic task offloading

To measure the performance of secure data communication in *TCFDL*, we perform the experiment under Case 3. As mentioned in Section 3, transmitted task data is processed by the TEE proxy in *TCFDL*. In the HSE scheme, the data is decrypted and then encrypted by the TEE proxy with the remote TEE proxy's AES key. Therefore, the data will be three times of symmetric encryption, as well as three times of symmetric decryption. In the PRE scheme, hybrid encryption is applied that only the AES key will be re-encrypted. Therefore, only one symmetric encryption and one symmetric decryption, plus two times of asymmetric encryption, are applied. The experimental results are shown in Fig. 7.

From Fig. 7, it can be seen that the transmission latency is higher than the one without security schemes. The reason is that the data needs to be processed by TEE proxies before it is published to the framework. Among two secure data transmission schemes, the PRE scheme performs better due to lower computational overhead, which only uses one multiplication on the elliptic curve and avoids data encryption/decryption on TEE proxy. In addition, the computational overhead of the HSE scheme increase as the data size increases. For

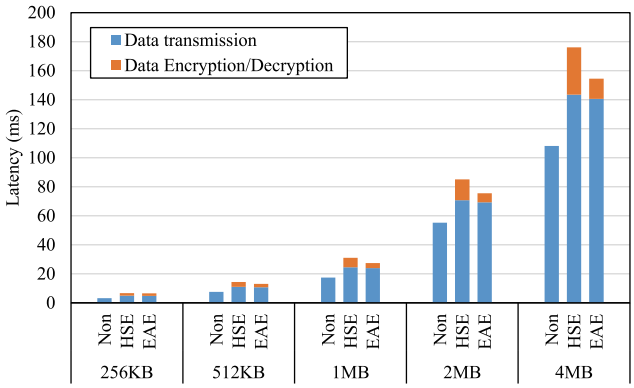


Fig. 7. Latency for data transmission and data encryption.

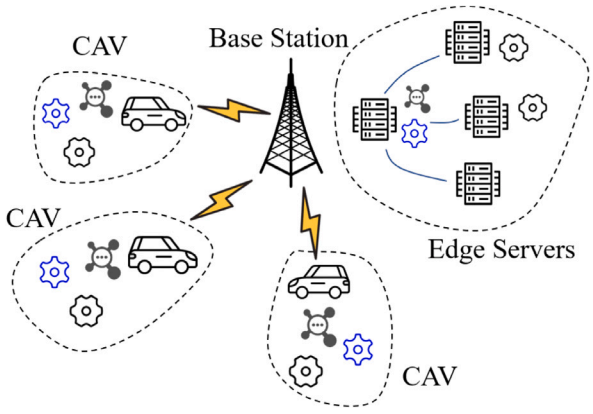


Fig. 8. A collaborative scenario for CAVs.

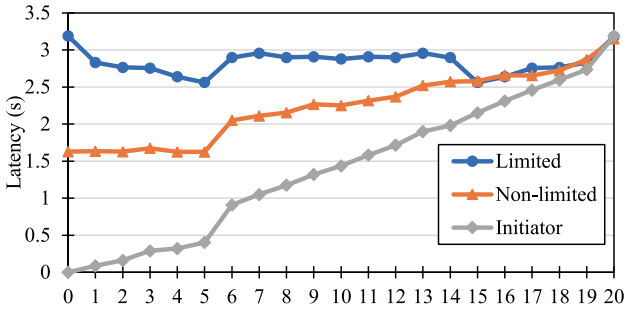


Fig. 9. Latency in collaboration with different edge servers.

example, they are 30 μs with 1 KB data and 490 μs with 256 KB data, respectively. However, the computation overhead is fixed to 200 μs.

6. Case study

In this section, we take the collaboration in the CAV scenario as a case study to demonstrate the functionality of *TCFDL*. One goal and contribution of *TCFDL* is that it supports large-scale AI model inference in TEE utilizing collaboration. As shown in Fig. 8, the framework in one CAV could offload tasks to the frameworks in the base station and other collaborative CAVs. All data is encrypted using the EAE approach to ensure the security of data transmission.

In the experiment, we set up two desktops as two CAVs, and two cases of base stations: (1) limited computing resources, which only consists of one desktop; and (2) non-limited computing resources, which has four desktops. All desktops have the same hardware as the previous experiments. To enable deep learning in TEE, we use Anakin as the underlying deep learning framework. In order to simulate future large-scale model inference in IoTs, we extend a popular model, MobileNet, by repeating part of layers, to 20x large, the number of parameters of which is still less than VGG16 model. The modified model is partitioned so that they could run on different nodes to simulate a collaboration, including CAVs and servers in the base station. In the limited computing resource case, the model offloaded to the base station cannot be partitioned again. It is because only one desktop in the cluster, which means the base station cannot scale more TEE instances in the cluster. In the non-limited computing resource case, the model offloaded to the base station will be partitioned into 4 parts inferred in four TEE instances scaled by TCFDL on 4 desktop, to make sure the partitioned model will not lead to a page-switching in Intel SGX with low performance.

Fig. 9 illustrates the response time in collaboration. The x-axis refers to the times of the model running in the CAV compared to the original MobileNet. The y-axis refers to the latency of obtaining the inference result from collaborative nodes. Besides, the latency of inferring such a model in the CAV is also shown in the figure, labeled as Initiator. The results show that collaboration could reduce latency. In addition, with the non-limited computing resources in the base station, the base station could partition and offload sub-models to its local worker nodes, which does not lead to a slowdown in computing. However, when the model offloaded to the edge servers is only with 5x MobileNet, the latencies are similar. It is because the offloaded model will not be partitioned in both the limited case and non-limited case.

7. Discussion

In this section, we will discuss the security of *TCFDL*, in terms of computation security and data transmission security, as well as overhead and limitations of *TCFDL*.

7.1. Security

Computation Security. TEE technology's security properties can guarantee the computational security of a function instance. The user function program can also verify the security of the framework through local attestation and obtain a certificate to prove its own security. Thus, collaborative function instances can attest to each other via certificate verifying. Finally, a trusted collaboration can be established, supported by TEE.

Transmission Security. To secure data in transmission, two approaches are proposed in this paper. In the HSE scheme, the data will be decrypted in the TEE proxy, which could be verified to keep confidential information such as plaintext and decryption keys are not accessible to the framework. To this end, data security in the transmission is guaranteed under the first security model. In the EVE scheme, the data is not decrypted on the TEE proxy, but is directly re-encrypted. Also, the re-encryption key cannot be used to decrypt the data, which has been proved by [36]. Thus, the EVE scheme can be secured under both the first and the second security models. In addition, the EVE scheme can guarantee security without the TEE proxy, while the message queue module can execute its re-encryption operation.

7.2. Limitations and future work

Message based Communication. In our *TCFDL*, all communication is message-based. On the one hand, this messaging paradigm is already used in many traditional IoT systems, *e.g.*, message queuing telemetry transport based IoT applications and systems. On the other hand, it is also used in existing FaaS platforms, such as OpenFaaS. We verify the feasibility of message-based collaborative computation in IoTs through a case study based on the implemented prototype.

However, the experiments show that our framework has a high additional overhead for across-cluster communication. Actually, to keep the maintainability and extensibility of our framework, we do not modify the underlying message queue system and implement our message queue module imitating the implementation of NATS Streaming, which also uses a NATS client to implement reliable transmission. Thus, we use a NATS Streaming client to handle messages in Pub/Sub and a NATS client to forward Request/Reply messages. To improve the performance of *TCFDL*, we will modify the message queuing module to avoid such forwarding.

Scalability. The TCFDL framework could scale instances by monitoring the length of unprocessed message queue, and launch an new instance in a WNode that have enough resource. However, in our implementation, multiple TEE instances cannot be scheduled in the same WNode to avoid out-of-memory TEE hardware. It is because the memory usage and performance relationships of TEE instances vary across hardware implementations, *i.e.,* ARM TrustZone and Intel SGX.

Therefore, as future work, we intend to first incorporate the usage of TEE hardware into the monitoring metrics, and then explore the memory-performance relationship to propose corresponding scheduling algorithms to ensure the efficiency of TEE hardware usage within the cluster.

8. Related works

8.1. Edge collaborative framework

In the field of edge computing, a number of frameworks have been proposed [37–39]. Satyanarayanan et al. [37] proposed Cloudlet as an early framework supporting edge computing. The servers located at the edge of the network provide the runtime of virtual machines, so that computing tasks can be offloaded to these edge servers. Amento et al. [40] proposed FocusStack, a location-aware hybrid edge-cloud system that provides the ability to extend computation to various edge devices (*e.g.,* drones, vehicles). Mortazavi et al. [41] proposed the CloudPath computing platform, which enables one task processed on the nodes from devices to the cloud. Zhang et al. [13] proposed an edge-cloud collaborative framework, Firework, and Zhang et al. [38] implemented a collaborative edge video analytics application, AMBER Alert Assistant, based on the Firework framework. Wang et al. [42] proposed MobileEdge, an in-vehicle collaborative computing platform that can migrate applications based on OpenCL kernels and Tensorflow models and then offload task data to process. Although it is proposed to deal with the collaborative computing problem in CAVs, it also can be extended to the edge-cloud collaborative environment. However, the above frameworks aim to deal with the basic problem of collaborative computing. They cannot provide a trusted environment for collaborative computing and dynamical task offloading with secure data transmission.

Some frameworks in the industry can also support collaborative computing in an edge-cloud environment. For example, Kubernetes [43], a well-known container orchestrator, has also released a lightweight version, KubeEdge, for edge computing. In addition, AWS Greengrass [44] and Apache Edgeent are also introduced as platforms supporting edge computing. However, these industry frameworks only provide the collaboration between devices and one or several fixed edge/cloud nodes, thus cannot provide a flexible collaboration at the edge.

8.2. TEE-enabled framework

With the introduction of the Trusted Execution Environment technology, the concept of confidential computing was introduced. Data can be secured in an absolutely trusted environment, and the data is handled explicitly without the fear of other applications being informed of the computation. At the same time, some TEE-enhanced computing systems and platforms were proposed. However, they only provide collaboration between two nodes. SecureStreams [45] utilizes Lua script to process streaming data in TEE. Based on the migration of Lua scripts, the applications could be migrated. In addition, it also provides an end-to-end secure data transmission. However, as mentioned before, it only provides a collaboration between two nodes, and also cannot support dynamic task offloading in a mobile environment. Similarly, StreamBox-TZ [26] also cannot provides a flexible collaboration.

Considering the edge computing environment, Vaucher et al. [46] designed a container scheduling plug-in for the KubeEdge scheduler that supports the use of SGX devices, monitors container usage of Intel SGX devices, and implements different scheduling algorithms. However, like the KubeEdge scheduler, it does not support collaborative computing and secure data communication mechanisms.

9. Conclusion

In this paper, we have investigated providing collaborative computing with security supports in a dynamic environment, in terms of computation and data transmission, for future AI-enabled IoTs. As a result, we proposed and implemented a trusted collaborative framework enhanced by hardware-assistant trusted execution environment technology. The proposed framework enables the collaboration between IoT devices, edge nodes, and clouds, and supports dynamic task offloading for mobile environments, such as connected and autonomous vehicles. In addition, to secure data in dynamic task offloading, we proposed two secure data transmission approaches in the framework. We implemented a prototype for the proposed framework and evaluated the performance. The experiment results show that our framework provides flexible and dynamic collaboration with low overhead. The case study of the collaboration in CAVs demonstrates that the framework can effectively support collaborative edge intelligence for future AI-enabled IoTs.

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.

Acknowledgments

The work was supported by the National Natural Science Foundation of China (No. 61872001, No. 6191101332, No. U1936220), the Open Fund of Key Laboratory of Embedded System and Service Computing (Tongji University), Ministry of Education, China (No. ESSCKF2018-03), the Open Fund for Discipline Construction, Institute of Physical Science and Information Technology, Anhui University, China and the Excellent Talent Project of Anhui University, China. The authors are very grateful to the anonymous referees for their detailed comments and suggestions regarding this paper.

References

[1] Q. Zhang, H. Sun, X. Wu, H. Zhong, Edge video analytics for public safety: A review, Proc. IEEE 107 (8) (2019) 1675–1696, http://dx.doi.org/10.1109/JPROC.2019.2925910, https://ieeexplore.ieee.org/document/8781894/.

[2] H. Li, K. Ota, M. Dong, Learning IoT in edge: Deep learning for the internet of things with edge computing, IEEE Netw. 32 (1) (2018) 96–101, http://dx.doi.org/10.1109/MNET.2018.1700202.

[3] Ericsson, Ericsson mobility report, 2020, https://www.ericsson.com/4adc87/assets/local/mobility-report/documents/2020/november-2020-ericsson-mobility-report.pdf.

[4] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, Deep learning for IoT big data and streaming analytics: A survey, IEEE Commun. Surv. Tutor. 20 (4) (2018) 2923–2960, http://dx.doi.org/10.1109/COMST.2018.2844341.

[5] J. Wang, J. Hu, G. Min, A.Y. Zomaya, N. Georgalas, Fast adaptive task offloading in edge computing based on meta reinforcement learning, IEEE Trans. Parallel Distrib. Syst. 32 (1) (2021) 242–253, http://dx.doi.org/10.1109/TPDS.2020.3014896.

[6] N. Luong, D.T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, D.I. Kim, Applications of deep reinforcement learning in communications and networking: A survey, IEEE Commun. Surv. Tutor. 21 (4) (2019) 3133–3174, http://dx.doi.org/10.1109/COMST.2019.2916583.

[7] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J.P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, J. Syst. Archit. 98 (2019) 289–330, http://dx.doi.org/10.1016/j.sysarc.2019.02.009, http://www.sciencedirect.com/science/article/pii/S1383762118306349.

[8] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, W. Shi, Computing systems for autonomous driving: State-of-the-art and challenges, IEEE Internet Things J. (2020) 1, http://dx.doi.org/10.1109/JIOT.2020.3043716.

[9] S. Grigorescu, B. Trasnea, T. Cocias, G. Macesanu, A survey of deep learning techniques for autonomous driving, J. Field Robotics 37 (3) (2020) 362–386.

[10] A.E. Sallab, M. Abdou, E. Perot, S. Yogamani, End-to-end deep reinforcement learning for lane keeping assist, 2016, arXiv:1612.04340.

[11] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646, http://dx.doi.org/10.1109/JIOT.2016.2579198.

[12] Q. He, C. Wang, G. Cui, B. Li, R. Zhou, Q. Zhou, Y. Xiang, H. Jin, Y. Yang, A game-theoretical approach for mitigatingedge ddos attack, IEEE Trans. Dependable Secure Comput. (2021) 1, http://dx.doi.org/10.1109/TDSC.2021.3055559.

[13] Q. Zhang, Q. Zhang, W. Shi, H. Zhong, Firework: Data processing and sharing for hybrid cloud-edge analytics, IEEE Trans. Parallel Distrib. Syst. 29 (9) (2018) 2004–2017, http://dx.doi.org/10.1109/TPDS.2018.2812177, https://ieeexplore.ieee.org/document/8306827/.

[14] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, N. Georgalas, Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning, IEEE Commun. Mag. 57 (5) (2019) 64–69, http://dx.doi.org/10.1109/MCOM.2019.1800971.

[15] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, Y. Yang, Auditing cache data integrity in the edge computing environment, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2021) 1210–1223, http://dx.doi.org/10.1109/TPDS.2020.3043755.

[16] Q. Zhang, Z. Yu, W. Shi, H. Zhong, Demo abstract: EVAPS: Edge video analysis for public safety, in: Proceedings - 1st IEEE/ACM Symposium on Edge Computing, SEC 2016, IEEE, 2016, pp. 121–122, http://dx.doi.org/10.1109/SEC.2016.30, http://ieeexplore.ieee.org/document/7774697/.

[17] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, S. Sinha, Real-time video analytics: The killer app for edge computing, Computer 50 (10) (2017) 58–67, http://dx.doi.org/10.1109/MC.2017.3641638.

[18] J. Xu, D. Zhang, L. Liu, X. Li, Dynamic authentication for cross-realm SOA-based business processes, IEEE Trans. Serv. Comput. 5 (1) (2012) 20–32, http://dx.doi.org/10.1109/TSC.2010.33.

[19] F. Ahmad, F. Kurugollu, C.A. Kerrache, S. Sezer, L. Liu, NOTRINO: a novel hybrid trust management scheme for internet-of-vehicles, IEEE Trans. Veh. Technol. (2021) 1, http://dx.doi.org/10.1109/TVT.2021.3049189.

[20] J. Cui, F. Wang, Q. Zhang, Y. Xu, H. Zhong, An anonymous message authentication scheme for semi-trusted edge-enabled iIoT, IEEE Trans. Ind. Electron. (2020) 1, http://dx.doi.org/10.1109/TIE.2020.3039227.

[21] M. Naehrig, K. Lauter, V. Vaikuntanathan, Can homomorphic encryption be practical? in: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, in: CCSW '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 113–124, http://dx.doi.org/10.1145/2046660.2046682, https://doi.org/10.1145/2046660.2046682.

[22] S. Sinha Roy, F. Turan, K. Jarvinen, F. Vercauteren, I. Verbauwhede, FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data, in: 2019 IEEE International Symposium on High Performance Computer Architecture, HPCA, 2019, pp. 387–398. http://dx.doi.org/10.1109/HPCA.2019.00052.

[23] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, J. Wernsing, Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy, in: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Vol. 48, in: ICML'16, JMLR.org, 2016, pp. 201–210.

[24] M. Sabt, M. Achemlal, A. Bouabdallah, Trusted execution environment: what it is, and what it is not, in: 2015 IEEE Trustcom/BigDataSE/ISPA, 1, IEEE, 2015, pp. 57–64.

[25] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, C. Rozas, Intel® software guard extensions (intel® SGX) support for dynamic memory management inside an enclave, in: Proceedings of the Hardware and Architectural Support for Security and Privacy 2016, in: HASP 2016, Association for Computing Machinery, New York, NY, USA, 2016, https://doi.org/10.1145/2948618.2954331.

[26] H. Park, S. Zhai, L. Lu, F.X. Lin, Streambox-TZ: Secure stream analytics at the edge with trustzone, in: 2019 USENIX Annual Technical Conference (USENIX ATC 19), USENIX Association, Renton, WA, 2019, pp. 537–554, https://www.usenix.org/conference/atc19/presentation/park-heejin.

[27] Google, Introducing asylo: an open source framework for confidential computing, 2020, https://cloud.google.com/blog/products/gcp/introducing-asylo-an-open-source-framework-for-confidential-computing.

[28] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, S. Yan, Occlum: Secure and efficient multitasking inside a single enclave of intel SGX, in: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, in: ASPLOS '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 955–970, http://dx.doi.org/10.1145/3373376.3378469, https://doi.org/10.1145/3373376.3378469.

[29] Q. Zhang, Y. Wang, X. Zhang, L. Liu, X. Wu, W. Shi, H. Zhong, Openvdap: An open vehicular data analytics platform for CAVs, in: Proceedings - International Conference on Distributed Computing Systems, 2018-July, IEEE, 2018, pp. 1310–1320, http://dx.doi.org/10.1109/ICDCS.2018.00131, https://ieeexplore.ieee.org/document/8416394/.

[30] Connected cars autonomous vehicles survey, 2018, https://www.foley.com/files/uploads/2017-Connected-Cars-Survey-Report.pdf. (Accessed 2018).

[31] NATS, NATS - open source messaging system, 2017, https://nats.io/.

[32] NATS, NATS streaming server, 2020, https://github.com/nats-io/nats-streaming-server.

[33] Redis Labs, Redis, 2020, https://redis.io/.

[34] A. Valialkin, Fast HTTP implementation for Go, 2020, https://github.com/valyala/fasthttp.

[35] Q. Zhang, H. Zhong, J. Wu, W. Shi, How edge computing and initial congestion window affect latency of web-based services: Early experiences with baidu? in: Proceedings - 2018 3rd ACM/IEEE Symposium on Edge Computing, SEC 2018, IEEE, 2018, pp. 393–398, http://dx.doi.org/10.1109/SEC.2018.00052, https://ieeexplore.ieee.org/document/8567697/.

[36] M. Blaze, G. Bleumer, M. Strauss, Divertible protocols and atomic proxy cryptography, in: K. Nyberg (Ed.), Advances in Cryptology — EUROCRYPT'98, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 127–144.

[37] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for VM-based cloudlets in mobile computing, IEEE Perv. Comput. 8 (4) (2009) 14–23.

[38] Q. Zhang, Q. Zhang, W. Shi, H. Zhong, Distributed collaborative execution on the edges and its application to AMBER alerts, IEEE Internet Things J. 5 (5) (2018) 3580–3593, http://dx.doi.org/10.1109/JIOT.2018.2845898.

[39] L. Yuan, Q. He, S. Tan, L. Bo, Y. Jiangshan, F. Chen, H. Jin, Y. Yang, Coopedge: A decentralized blockchain-based platform for cooperative edge computing, in: Proceedings of the Web Conference 2021, in: WWW '21, Association for Computing Machinery, New York, NY, USA, 2021, http://dx.doi.org/10.1145/3442381.3449994, https://doi.org/10.1145/3442381.3449994.

[40] B. Amento, B. Balasubramanian, R.J. Hall, K. Joshi, G. Jung, K.H. Purdy, Focusstack: Orchestrating edge clouds using location-based focus of attention, in: Proceedings - 1st IEEE/ACM Symposium on Edge Computing, SEC 2016, Institute of Electrical and Electronics Engineers Inc., 2016, pp. 179–191, http://dx.doi.org/10.1109/SEC.2016.22.

[41] S.H. Mortazavi, M. Salehe, C.S. Gomes, C. Phillips, E. De Lara, Cloudpath: A multi-tier cloud computing framework, in: 2017 2nd ACM/IEEE Symposium on Edge Computing, SEC 2017, Association for Computing Machinery, Inc, New York, NY, USA, 2017, pp. 1–13, http://dx.doi.org/10.1145/3132211.3134464, https://dl.acm.org/doi/10.1145/3132211.3134464.

[42] L. Wang, Q. Zhang, Y. Li, H. Zhong, W. Shi, MobileEdge: enhancing on-board vehicle computing units using mobile edges for CAVs, in: 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), 2019, pp. 470–479.

[43] D. Bernstein, Containers and cloud: From lxc to docker to kubernetes, IEEE Cloud Comput. 1 (3) (2014) 81–84.

[44] Amazon, AWS greengrass, 2020, https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html.

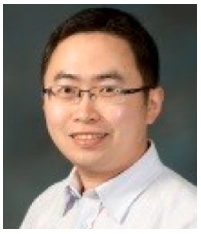
[45] A. Havet, R. Pires, P. Felber, M. Pasin, R. Rouvoy, V. Schiavoni, Securestreams: A reactive middleware framework for secure data stream processing, in: DEBS 2017 - Proceedings of the 11th ACM International Conference on Distributed Event-Based Systems, Association for Computing Machinery, Inc, New York, New York, USA, 2017, pp. 124–133, http://dx.doi.org/10.1145/3093742.3093927, http://dl.acm.org/citation.cfm?doid=3093742.3093927.

[46] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni, C. Fetzer, SGX-aware container orchestration for heterogeneous clusters, in: Proceedings - International Conference on Distributed Computing Systems, 2018-July, Institute of Electrical and Electronics Engineers Inc., 2018, pp. 730–741, http://dx.doi.org/10.1109/ICDCS.2018.00076.

Qingyang Zhang received the B. Eng. degree in computer science and technology from Anhui University, China in 2014, where he is currently pursuing the Ph.D. candidate. His research interest includes edge computing, computer systems, and security.

Hong Zhong was born in Anhui Province, China, in 1965. She received her Ph.D. degree in computer science from University of Science and Technology of China in 2005. She is currently a professor and Ph.D. supervisor of the School of Computer Science and Technology at Anhui University. Her research interests include applied cryptography, IoT security, vehicular ad hoc network, cloud computing security and software-defined networking (SDN). She has over 120 scientific publications in reputable journals (e.g. IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Vehicular Technology, IEEE Transactions on Intelligent Transportation Systems, IEEE Transactions on Network and Service Management, IEEE Transactions on Big Data and IEEE Internet of Things Journal), academic books and international conferences.

Weisong Shi is a Charles H. Gershenson Distinguished Faculty Fellow and a professor of Computer Science at Wayne State University. His research interests include Edge Computing, Computer Systems, energy-efficiency, and wireless health. He received his BS from Xidian University in 1995, and Ph.D. from Chinese Academy of Sciences in 2000, both in Computer Engineering. He is a recipient of National Outstanding Ph.D. dissertation award of China and the NSF CAREER award. He is an IEEE Fellow and ACM Distinguished Scientist.



Lu Liu is the Professor of Informatics and Head of School of Informatics in the University of Leicester, UK. Prof Liu received the Ph.D. degree from University of Surrey, UK and M.Sc. in Data Communication Systems from Brunel University, UK. Prof Liu's research interests are in areas of cloud computing, service computing, computer networks and peer-to-peer networking. He is a Fellow of British Computer Society (BCS).