MPC-as-a-Service: A Customizable Management Protocol for Running Multi-party Computation on IoT Devices

Oscar G. Bautista and Kemal Akkaya
Department of Electrical and Computer Engineering
Florida International University, Miami, FL, USA
Email: {obaut004, kakkaya}@fiu.edu

Abstract—Techniques to perform computations without disclosing the input values have notably improved in the last decade. One such technology, called Secure Multiparty Computation (MPC), where two or more computation nodes hold secret pieces of private data and jointly execute a protocol to obtain a function output, has proven effective for preserving privacy in many applications (e.g., distributed signing, financial scores, machine learning, and more). Nonetheless, in many cases, the data source and computation nodes are often assumed to be the same, with the existence of a manually preconfigured network before they start the computation. This challenge is typical of many IoT applications where the IoT devices need to collaborate using MPC but do not have the resources, and thus outsource the tasks to powerful MPC nodes. Nonetheless, in such a scenario, the IoT devices do not know the MPC nodes, and vice-versa to manage the overall process. To fill this gap, we propose an MPC management protocol that automates the registration and authentication of a group of clients (i.e., sources and consumers of data) and MPC servers (the private computation providers), the requesting of MPC jobs, and receiving the results thereafter. Our experiments over a cloud environment demonstrate the first protocol that efficiently and securely automates the management of MPC systems on many use cases, which would otherwise take considerable time and effort.

Index Terms—MPC, MPC-as-a-service, multiparty computation, management protocol, network authentication, encryption

I. Introduction

The ability to do computation using private data is a critical technology in the privacy era, where protecting personal or sensitive data is mandated in increasing number of applications. One such privacy-preserving computation technology is Secure Multiparty Computation (MPC) [1], which performs computation on data secret-shared among n computation nodes such that at least $t <= n$ nodes are required to reconstruct the underlying private value (e.g., an input to or output of the computation).

While MPC is suitable for cases where privacy-sensitive data is owned by or stored in different locations or organizations, critical challenges hurdle the broad implementation of this technology. For instance, MPC protocols, in general, have several security parameters requiring certain knowledge about the specific computation mechanism to set up the computation system adequately in advance. Also, the minimum computing resources needed for an MPC node to complete a particular task may be more than those of general-purpose computers [2], [3], and specially of resource-constrained IoT devices. Some good examples are a group of drones collecting terrain information, or healthcare IoT systems, where devices for remote patient care collect sensitive information and require high processing computations. In any case, they cannot perform the secure computation because of significant energy spending, computation and communication limitations. Therefore they need to outsource this task to more powerful nodes.

Although secure solutions for outsourcing the data and the private computation to dedicated MPC nodes already exist [4], they assume a preexisting network with properly secured channels interconnecting all the nodes. This assumption bypasses the challenge of distributing the connection information of all participants in the first place. Additionally, the interested entities have to deploy the computation nodes themselves, incurring additional costs. While the gained benefit of privacy can justify certain cost increases, there are cases where a private computation is only required occasionally, which leads to under-utilization of the computational resources. MPC-as-a-Service (MPCaaS) [5] is proposed to solve these challenges, which means MPC services can be accessed on-demand.

While general architecture and components of an MPCaaS system, which may vary slightly depending on the specific needs, are described, there exists no protocol that broadly describes managing an MPC system in the outsourced setting (e.g., IoT devices are heavily involved), including MPCaaS. Specifically, the basic needs of an MPC deployment that a management protocol can streamline are: 1) Registration and authentication of devices, and 2) Setting up channels among MPC servers and clients.

We aim to fill this gap by proposing and defining an MPC management protocol that covers all stages, from participant registration with an MPC system, to the MPC execution, and successful output delivery. We assume a general architecture where the private or sensitive data resides in a group of clients, and a group of MPC servers executes the secure computation. This group of MPC servers is a subset of a larger pool of MPC servers (e.g., servers hosted on different cloud service providers (CSP) for resilience and security) categorized by attributes such as geographical location and computation resources, among others. This categorization enables a better quality of service when possible by fulfilling the client's requirement first and

then randomizing the final selection, assuming enough servers meet the requirements.

We offer a comprehensive framework instantiating a proposed set of protocols for participant registration (i.e., client or server), Kerberos-like authentication [6], and MPC job orchestration. This framework suits the need of organizations that wants to set up their MPC system while temporarily incorporating computation power from other stakeholders or third parties. Our experiments with an actual protocol implementation demonstrate how quick and secure is to set up an MPC system, eliminating the tedious and time-consuming task of manually setting up secure channel connections, transferring encryption keys, etc.

The rest of the paper is organized as follows: Section II overviews previous work on MPC management for IoT environments. Section III presents background information. Section IV describes the main components of a general MPC architecture in the outsourced setting. Section V describes in detail our proposed protocols. Section VI discusses our experiment results, and finally, we present our conclusion in Section VII.

II. Related Work

M. von Maltitz et al. [7] proposed a management framework including the initial setup and execution orchestration where the entities–called peers–holding the data (e.g., sensor platforms) also perform the computation. Additionally, one of those peers acts as the gateway coordinating the whole process. Such architecture is atypical in real-life as these coordination and management tasks add to the demand of resources from IoT devices. Finally, it lacks the description of the protocols that accomplish each functionality. Our proposed protocol allows IoT devices to securely outsource the sensitive data and the private computation to more powerful nodes, releasing them of computational burden and establishing secure communication without any previous knowledge of the MPC server's identity.

H. Gao et al. [8] proposed a blockchain-based MPC scheme where users (i.e., the clients) negotiate a private computation contract over the blockchain, and a network of MPC servers provides the computation service. Although their focus is on MPC fairness and robustness, their architecture is similar to ours in that clients outsource the data and the computation to more powerful nodes. Nonetheless, the users/MPC contract selects MPC servers based essentially on their reputation (i.e., assumes computation power and location, among other attributes, are irrelevant). Furthermore, the scheme relies on a specific MPC protocol, and it is not clear how the privacy of the inputs is protected when outsourced to the MPC servers.

Barak et al. [9] present an MPC system for low bandwidth and weak participants with low communication complexity and small fields (e.g., the size of the secret shares) so that IoT devices can directly run the MPC protocol. The proposed system includes an administrative component that sets up an MPC computation and invites participants to enroll using Google or Facebook authentication. Thus, the system does not provide user identity anonymity. Our proposed MPC management protocol is, in a sense, an alternative to the administrative component of the cited work that uses device-based network authentication and enables automation of the MPC setup and execution orchestration for any MPC protocol. It also enables data sources/owners to securely outsource their sensitive information to powerful MPC servers and receive the results in the end.

III. Background Information

A. Secure Multiparty Computation

In secure multiparty computation [1], two or more interconnected parties which hold pieces (i.e., secret shares) of each private value jointly evaluate a function without knowing the actual value of the inputs and outputs. The MPC protocol relies on a secret-sharing scheme to achieve secrecy. For instance, let x be a private input and $\langle . \rangle$ the operator denoting secret-shared form. Then $\langle x \rangle := (x_1, x_2, ..., x_n)$, where n is the number of parties performing the secure computation.

B. Kerberos Protocol

Kerberos [6] is a computer network authentication protocol where users and services authenticate to each other over an insecure network. The basic components of the Kerberos ecosystem are the Authentication Server (AS), the Ticket Granting Server (TGS), the Service, and the User. A user request access to a service by sending a request to AS, which verifies that the user exists, returns a response that includes a ticket-granting session key and a ticket-granting ticket (TGT). The first message is encrypted with the user's private key, and the second one is encrypted with the TGS's private key. Next, the user sends a couple of messages to TGS requesting access to the service. The user's request is encrypted with the session key generated by AS. The TGS recovers the same session key included in the TGT by the AS so that the user and the TGS can communicate securely. At this point, the TGS verifies that the service exists and replies to the user with two messages: a service session key encrypted with the ticket-granting session key and a service ticket (ST) which also contains the service session key and is encrypted with the service's secret key. Finally, the user authenticates to the service repeating the same procedure but using the newly generated keys. Since the user and the service now have a common service session key, they can communicate securely.

IV. MPC Architecture in the Outsourced Setting

We assume an MPC scenario as depicted in Fig. 1, where entities (i.e., clients) that own private (i.e., personal or confidential) data differ from the computing parties. Therefore, they need to outsource the data (e.g., $\langle x \rangle, \langle y \rangle$, etc.) and the computation securely to MPC servers or parties. In this section, we describe the components of such general MPC system architecture and provide insights about how it applies to different scenarios including multiple IoT nodes sensing and transmitting sensitive information.

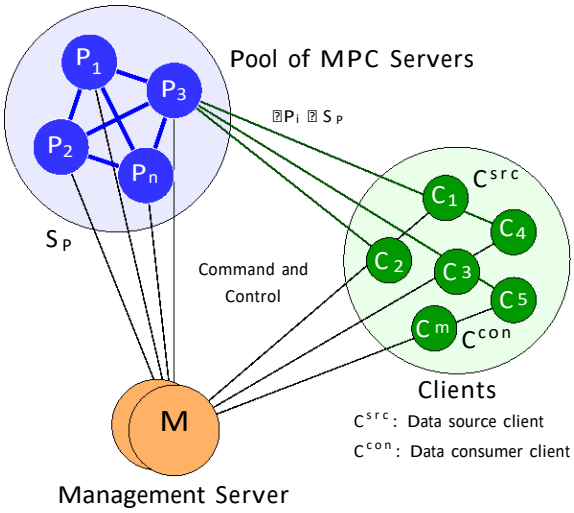

Fig. 1. General Architecture of an MPC system in the Outsourced Setting

A. MPC Servers

These are the nodes that execute the private computation. The MPC servers hold specific attributes that help categorize them, such as computation resources (i.e., CPUs, RAM), geographical location, and other characteristics which may impact the performance. For example, some MPC jobs require evaluating large amounts of data or computing more complex functions, which makes it preferable to select more powerful computation servers. Generally, MPC protocols are semi-synchronous. They execute in rounds where each server receives all messages from their peers for the current round before moving on to the next round. This means that if a server is slower than the others, it will dictate the speed of the MPC execution. At the same time, faster servers will underutilize their resources by exhibiting more significant idle times. For this reason, the selection of servers should favor servers with similar capabilities to carry out a specific task.

Any stakeholders associated with the privacy-preserving application can deploy MPC servers. Additionally, third parties can also offer their computation resources for a fee. A client may request servers in a specific country or region to perform a particular MPC job. This might be useful in cases where one needs to comply with data processing regulations [10]. There is, nonetheless, a performance-based reason to prefer MPC servers in the same region. Having regional deployments helps minimize the communication delay among the computation nodes and reduce the total execution time [3], [11], [12].

B. Clients

These entities or nodes provide the private input or get the output of the computation. We refer to those nodes as source clients (C^{src}) and consumer clients (C^{con}), respectively. An C^{src} is commonly characterized for having limited resources, hence, they need to outsource their data and secure computation. For instance, a source client could be a cluster-head in an IoT sensor networks, or one the IoT devices itself in healthcare for remote patient monitoring. In other applications, the source and consumer client can be the same physical device, such as a server holding a group of pictures from which they want to find specific objects using an ML-trained model offered by another organization and receive back the results of the automated object identification. In this example, one participant wants to keep its proprietary ML model private, and the other wants the same with the content of the pictures.

C. Management Server

This server, referred to as M, oversees the whole MPC ecosystem. Some of its functions are registering new participants (i.e., MPC servers or clients) and authenticating and authorizing them to set up adequately secured network connections among them before an MPC job execution. It also enables monitoring of participants and thus could intervene when needed (e.g., failure recovery). The management server IP is the only address known by the participants at the beginning. In practice, the management server role can be implemented using Anycast [13] to address a group of geographically distributed servers using a single IP address while eliminating the single point of failure. The stakeholders requiring a private computation can deploy these servers as they will maintain a database with each participant's authentication information. Note that the management servers do not learn anything about the private inputs. As we describe in the next section, the secret input shares are transmitted directly from clients to MPC servers.

V. MPC Management Protocol Suite

This section describes the protocols that collectively realize the MPC management framework. We also briefly discuss how the design considerations relate to different applications, making the proposed framework flexible enough to fit most currently known use cases.

As with any service, the first step for a node to participate in any capacity is registering with the system and agreeing on an identity that will get authorization for future requests. We differentiate between the registration procedure for clients and MPC servers. Then, the authentication of both types of participants is very similar. Finally, we describe a high level (application) protocol for MPC job request and execution details.

A. MPC Server Registration Protocol

Recall that third parties can deploy the MPC servers in exchange for a fee. Additionally, MPC servers are untrusted by default. They participate in the MPC ecosystem because of their computation resources without providing data. Therefore, they do not necessarily need a public identity. We propose that the MPC system can identify and authenticate any single MPC server at any time using the identity created during the registration.

The MPC server registration protocol consists of two communication rounds. In the first round, the MPC server sends a registration request along with its public key to the management server, which responds with the newly created identity (i.e., a name and a symmetric key) and a request to learn MPC server's

attributes. In the second round, the MPC server provides the requested attributes, which include at least its computational capabilities and geographical location, but may also include customized attributes such as the list of MPC protocols that it can execute. At the end of this registration process, the MPC server and the management server will share a symmetric key that will be used for authentication and to communicate securely and efficiently hereafter.

The management server also stores the MPC servers' IP addresses so that it has the option to allow only authentication requests from the IP address which was used by an MPC server during registration. Once registered, the MPC server initiates the authentication protocol (see Section V-C) and establishes a secure control channel with the management server.

B. Client Registration Protocol

The clients have a more restricted treatment when registering them with the MPC system. This is because there may be various attacks through fake clients registered in the system. For example, they could learn the output, which in some cases could be sensitive. In another case, a fake client could provide false data and and affect the computation results' accuracy.

When it comes to client registration and association to an MPC application, the proposed MPC management framework uses an application ID (A^{id}) that associates a group of clients and specific private data labels so that when a consumer client requests an MPC job, the MPC system knows which source clients are associated with such A^{id} and orchestrates the MPC accordingly.

We propose two methods to validate the registration request from clients to the MPC system: The first method assumes the availability of a set of authorization codes which the management server generates using the information provided by the IoT application's administrator. Such authorization codes can be distributed out-of-band and be optionally associated with a predefined device name or IP address. The second method provides a single authorization code to all clients. In this case, the management server is open to new registrations for the specific A^{id} during a particular period or for a maximum number of clients. The selected mechanism would depend on the use case particularities. The first one offers more control over the individual clients authorized and is suitable when we do not have direct control about when the client registration will happen, while the second one guarantees that after the initial registration, no additional node would get access as part of the same application.

The client registration protocol differs from the MPC server registration mainly in two aspects. First, it requires the client and server to run a key agreement protocol (e.g., Diffie-Hellman) that can be executed by resource-constrained devices, then they can share a symmetric key which the client will use to prove its identity. Second, client and server derive this key from a long string made concatenating the application ID, client name, and a password. This is done to be flexible on how the end IoT devices manage their credentials (e.g., local key storage or user login with a password). The design consideration for this protocol includes using only symmetric key cryptography, which allows for the registration of constrained-resource IoT devices. Nonetheless, in the case of more powerful devices (e.g., computers, externally powered remote devices) registering as clients, they can also use a variation of the protocol based on public-key infrastructure.

Protocol 1 MPC Server Authentication

Require: An MPC server P holding P^{name} and P^{smk}.

1: P sends an unencrypted connection request message to M. This message contains P^{name} if the IP address changed
2: M looks up the requester's IP address in its database and fetches P^{smk}
3: M sends a counter or similar challenge along with P^{name} in a message encrypted using P^{smk}
4: P decrypts the received message using its P^{smk} to recover the counter
5: P assembles a new message including a timestamp, the counter incremented, and encrypts it using P^{smk} and sends it to M
6: M decrypts the message, confirms the counter increment and verifies that timestamp is within a permissible range
7: If all is in order, M sends a confirmation message to P and the control connection is established.

C. Authentication Protocol

Registered participants authenticate to the management server establishing a secure channel to submit requests and receive responses (for clients) or orchestrate and monitor the MPC execution (for MPC servers). As indicated, during the registration, the participants get a unique name as their identifier, and they can authenticate to the management server using a challenge-response authentication mechanism (CRAM) along with the unique symmetric key they share to prevent replay attacks.

The authentication protocol is very similar for clients and MPC servers. The only difference is that clients can optionally derive the symmetric key from a password. Additionally, MPC servers can use the IP address as their identifier if the option to allow only known IP addresses is enabled. Protocol 1 shows the details of the authentication of MPC servers. The equivalent client authentication is essentially the same as mentioned and is not included due to space constraints.

D. MPC Request and Setup Protocol

Recall that MPC consists of a group of computation nodes executing a protocol jointly. Therefore, they need to set up a network (i.e., intranet) to exchange messages. Additionally, each source client needs to establish a direct secure connection between itself and every MPC server to securely distribute their shares, as shown in Fig. 1. Since the registration phase only enables connectivity between each participant and the management server, we need an automated MPC setup protocol to enable a participant to identify and establish secure communication with the rest.

Note that the communication setup is specific for each MPC job requested. Therefore, the management server carries it whenever a consumer client makes such a request. We propose

a new protocol whose details are shown in Protocol 2, and Fig. 2 shows the corresponding message exchange sequence.

Protocol 2 MPC Setup Protocol

Require: An MPC system comprising a pool of MPC servers $S_P = \{P_1, ..., P_n\}$ and source and consumer clients.

1: A consumer client C^{con} sends an encrypted message requesting a new MPC job with some specifications J^{parm}
2: M creates a Job Id J^{id} and proceeds to notify the related C^{src} about the data request.
3: Each C^{src} acknowledges the notification and responds with the size of the data available for the specific job.
4: M matches online P_i parameters with J^{parm}, and sends them a job request with the input data size.
5: Each P_i responds with their availability status and any other customized status codes to M.
6: M sends a message to C^{con} containing J^{id} along with the request confirmation or a counteroffer. The protocol exits if C^{con} rejects the counteroffer.
7: C^{con} sends M a message confirmation or aborting the MPC job request.
8: M distributes the IP addresses and identities (signature verification keys) among the MPC group for this J^{id}.
9: M creates and distributes to each C_j^{src}: (1) new symmetric session keys P_{i,C_j}^{ssk} for each $P_i - C_j^{src}$ connection; and (2) session tickets T_{P_i,C_j} containing the key from (1) and encrypted with P_i^{smk} that authenticates C_j^{src} to P_i.
10: Once M receives a confirmation of setup complete from each P_i, M replies commanding each P_i to start the MPC job execution.

It starts with the consumer client requesting some desired parameters, such as the number and location of the MPC servers, which influence the security and performance of the execution. Additionally, when the pool of MPC servers have a varying computation capacity, the servers are categorized which allows us to group those with similar computation resources. All servers in the computation perform mostly the same operations. Therefore, choosing servers that possess identical capabilities maximizes resource usage. The requester could also choose the specific MPC algorithm to execute. The deployment allows for more customization adding requirements or options as needed. Upon receiving the request in step 1, the management server executes step 2, verifying which source clients are online, and each available client responds with the size of the data they have in step 3. Next, the management server computes a match of the requirements versus the online MPC servers and sends out a round of reservation messages (step 4), to which they respond with their availability status (step 5). In step 6, the management server reports the configuration of MPC servers and source clients to the requester (i.e., the specific consumer client). This response is a counteroffer that could be less than the original request (e.g., more online servers need to meet the specific requirements). At this point, the requester can abort and try later or accept the counteroffer. In step 7, the requester confirms the job execution so that the management server shares the identities of the MPC servers among themselves (step 8) and generates and distributes credentials for each input client to authenticate to each MPC server (step 9).

We were inspired by the Kerberos [6] protocol to work with this specific architecture. Specifically, referring to the original protocol described in Section III-B, the TGS was removed, and the management server generates and distributes a Service Ticket for each client-MPC server pair. This is summarized as step 9 in Protocol 2. Finally, the MPC servers set up the network among themselves and notify the management server when they are ready to start carrying out the MPC job so that all can start simultaneously.

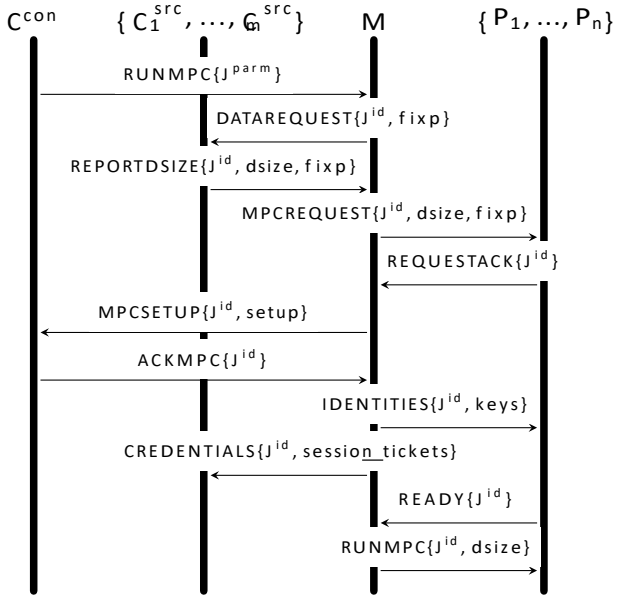

Fig. 2. MPC Setup Protocol Message Exchange

VI. Implementation and Experiments

We implemented and tested our MPC management framework in Python 3. The source code is available at our GitHub repository [14].

A. Experiment Setup

We conducted experiments locally (i.e., using the localhost and a WLAN) and over a WAN. We deployed three MPC servers on the cloud using three different Cloud Service Providers (CSP) (i.e., GCP, AWS, and Azure) using VMs with two vCPUs and 2 to 4 GB of RAM. We also set up six clients to measure the protocol execution time, assuming a distributed IoT deployment.

B. Participants Registration

In this experiment, we added new participants to the MPC framework. First, we conducted an experiment running the participants (i.e., MPC server and client) and the management server in the same host equipped with an Intel(R) Core i7-10750H processor, and measured the time it takes for each type of participant to register. Then we repeated the experiment running the participants on a different device inside the same WiFi network. This experiment gives us an idea of the portion of time due to network communication delay.

Table I shows the time measurement results for both types of participants in both experiments. In the case of the MPC server, the execution includes the time it takes to generate

a private-public key pair and the corresponding encryption and decryption operations as defined by the protocol. Note that in the case of client registration, the protocol uses only symmetric encryption, which is a design consideration suitable for constrained-resource devices.

TABLE I
PARTICIPANT REGISTRATION TIME

Participant Type	Running on Localhost	Over WiFi
MPC server	833 ms	881ms
Client	60 ms	111ms

C. MPC Network Setup Protocol

Once a consumer client makes and MPC job request, the management server begins executing Protocol 2. We ran experiments with two and three MPC servers and three to five data source clients in addition to the consumer client.

We measured the protocol execution time on the management server side to have a common clock reference. Specifically, referring to Protocol 2, we start the timer after receiving the message from step 1 and stop it at step 10.

We conducted experiments in two scenarios. First, with the source clients running over WAN, and second with the source clients deployed on a WiFi network in the same location as the management server. The MPC servers, the management server, and the consumer client have the same location in both scenarios. Table II shows the participants' locations in the WAN scenario (i.e., including the source clients). For the second scenario, the clients run from several wireless devices, including one Raspberry Pi 3B, which communicate through WiFi with the management server.

TABLE II
TYPE AND LOCATION OF PARTICIPANTS IN THE WAN SETTING

Type	CSP/Location	Type	CSP/Location
Mgmt. server	Local/Miami	Src. client 1	GCP/Los Angeles
MPC server 1	GCP/Iowa	Src. client 2	AWS/Ohio
MPC server 2	AWS/N.California	Src. client 3	Azure/Texas
MPC server 3	Azure/Virginia	Src. client 4	GCP/Salt Lake city
Consu. client	GCP/N.Virginia	Src. client 5	GCP/S.Carolina

Table III shows the time measured for both WiFi and WAN scenarios, where in each scenario, the delay variations changing the number of participants are minor. Furthermore, there is almost no difference for IoT devices (i.e., RPi over WiFi) to setup.

TABLE III
MPC REQUEST AND SETUP PROTOCOL RUN TIME

# of MPC Servers	# of Clients	Run time on WAN [ms]	Run time on WiFi [ms]
2	3	406	402
2	4	442	510
2	5	449	512
3	3	486	496
3	4	490	527
3	5	494	542

Although the main benefit of the MPC setup protocol is reducing time and effort before the actual computation while fulfilling the MPC system parameter requirements, its fast execution may become critical in applications requiring an autonomous setup for recurrent and short, secure computations. These experimental results show that our proposed protocol can meet those requirements.

VII. CONCLUSION

We proposed a comprehensive management framework for MPC in the outsourced setting (including MPC-as-a-Service) which integrates protocols for participant registration, Kerberos-like authentication, and MPC job orchestration. This framework saves time and effort for owners and administrators of IoT applications collaborating with other stakeholders while keeping their information private. Furthermore, our experiments on local and national deployments show that our framework sets up autonomously a secure MPC network in less than one second over an insecure network without any previous knowledge of the MPC servers identities.

ACKNOWLEDGMENT

This research was supported by the U.S. National Science Foundation, award number US-NSF-1663051.

REFERENCES

[1] Y. Lindell, "Secure multiparty computation," Commun. ACM, vol. 64, no. 1, p. 86–96, dec 2020.
[2] Z. Huang, W. jie Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure Two-Party deep neural network inference," in 31st USENIX Security Symposium (USENIX Security 22), Aug. 2022, pp. 809–826.
[3] S. Wagh, D. Gupta, and N. Chandran, "Securenn: Efficient and private neural network training," IACR Cryptol. ePrint Arch., p. 442, 2018.
[4] I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft, "Confidential benchmarking based on multiparty computation," in International Conference on Financial Cryptography and Data Security. Springer, 2016, pp. 169–187.
[5] P. B. Foundation, "Mpc techniques series, part 10: Mpc-as-a-service — the partisia blockchain infrastructure," 2021. [Online]. Available: https://medium.com/partisia-blockchain/mpc-techniques-series-part-10-mpc-as-a-service-the-partisia-blockchain-infrastructure-9b4833e77965
[6] C. Neuman, USC-ISI, T. Yu, S. Hartman, and K. Raeburn, "The kerberos network authentication service (v5)," RFC, 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4120
[7] M. von Maltitz et al., "A management framework for secure multiparty computation in dynamic environments," in NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–7.
[8] H. Gao, Z. Ma, S. Luo, and Z. Wang, "Bfr-mpc: A blockchain-based fair and robust multi-party computation scheme," IEEE Access, vol. 7, pp. 110 439–110 450, 2019.
[9] A. Barak, M. Hirt, L. Koskas, and Y. Lindell, "An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '18, 2018, p. 695–712.
[10] Intersoft Consulting, "General data protection regulation (gdpr)," 2023. [Online]. Available: https://gdpr-info.eu/
[11] O. G. Bautista and K. Akkaya, "Network-efficient pipelining-based secure multiparty computation for machine learning applications," in 2022 IEEE 47th Conference on Local Computer Networks (LCN), 2022, pp. 205–213.
[12] M. Byali, C. Hazay, A. Patra, and S. Singla, "Fast actively secure five-party computation with security beyond abort," ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1573–1590.
[13] J. Abley, A. Canada, and K. Lindqvist, "Operation of anycast services," RFC, 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4786
[14] O. G. Bautista, "Mpc management framework," GitHub, 2023. [Online]. Available: https://github.com/adwise-fiu/mpc-mgmt-cmdl