DOI:10.1145/3677390

Federated learning and confidential computing are not competing technologies.

BY JINNAN GUO, PETER PIETZUCH, ANDREW PAVERD, AND KAPIL VASWANI

Trustworthy AI Using Confidential Federated Learning

THE ARTIFICIAL INTELLIGENCE (AI) revolution is reshaping industries and transforming the way we live, work, and interact with technology. From AI chatbots and personalized recommendation systems to autonomous vehicles navigating city streets, AI-powered innovations are emerging everywhere. As businesses and organizations harness AI to streamline operations, optimize processes, and drive innovation, the potential for economic growth and societal advancement is immense.

Amid this rapid progress, however, it is critical to ensure AI's trustworthiness. Trustworthy AI systems must exhibit certain characteristics, such as reliability, fairness, transparency, accountability, and robustness. Only then can AI systems be depended upon to operate ethically and effectively without causing harm or discrimination.

A critical aspect of trustworthy AI is privacy. Training accurate machine-learning (ML) models often requires large, diverse, and representative datasets. While models can be trained exclusively using publicly available datasets in some domains, other scenarios require access to private data. For example, training models to make medical diagnoses may require sensitive patient data. Similarly, training models to detect fraudulent financial transactions requires detailed transaction data from financial institutions. Such data must be safeguarded from unauthorized access, manipulation, or misuse to maintain model integrity and prevent bias.

Consequently, there has been growing interest in privacy-preserving ML techniques, such as federated learning (FL).[17] FL is a distributed ML paradigm th ˜at enables training models across multiple clients holding local training data, without exchanging that data directly. In a typical FL setup, a central aggregator starts a training job by distributing an initial model to multiple clients. Each client trains the model locally on its dataset and computes updates to the model (also referred to as gradient updates). The clients then send their updates to the central aggregator, which aggregates these updates using a suitable aggregation function and updates its model. It then starts another epoch by sending the updated model to the clients, which perform local training. This process repeats until the mutually agreed-upon termination criteria are met (for example, the model converges to an acceptable loss value).

FL can be combined with differential privacy[7] to provide strong privacy guarantees.[24] In this setting, each cli-

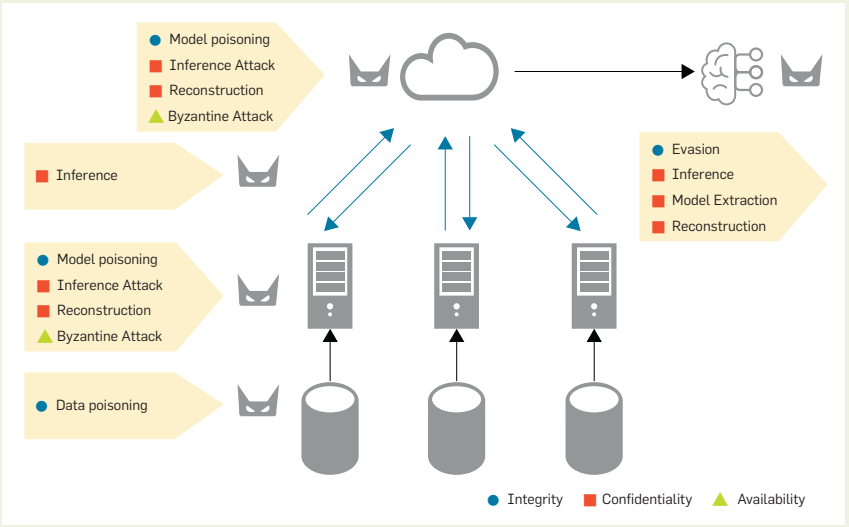
IMAGE BY LEAVE A TRACE

ent adds suitable noise to the model updates locally, based on a privacy budget, before sending the updates to the aggregator, which bounds the probability for the model to memorize individual points in the training dataset.

While FL prevents the flow of raw training data across trust domains, it introduces a new set of trust assumptions and security challenges. Clients participating in FL must trust a central aggregator to deliver safe code, include only trustworthy clients, follow the aggregation protocol, and use the model only for mutually agreed-on purposes. In addition, the aggregator must trust the clients to provide high-quality data, not tamper with the training protocol, and protect the model's intellectual property. These trust assumptions are often difficult to satisfy in the real world, especially in adversarial settings where clients may be compromised or collude to undermine the system's security and privacy guarantees. It is therefore unsurprising that many FL deployments have been found to be vulnerable to attacks, including model poisoning, data poisoning, and inference attacks[8,10,22] (see Figure 1). Attacks may be carried out by clients, aggregators, or outsiders, and can occur during model training or inference.

Many of these attacks can be attributed to the ability of malicious participants to violate the confidentiality or integrity of data and computation in their control (for example, by poisoning datasets or gradient updates to influence the model's behavior). These attacks are not limited to just the aggregators or clients at training time—attacks such as model extraction or

Figure 1. Attacks on federated learning systems.



reconstruction can be carried out by entities with API access to the trained model at inference time. Therefore, it is critical to protect all sensitive information throughout the lifecycle of FL jobs.

Another challenge in FL is transparency and accountability. Since, by definition, FL does not involve sharing training data directly, it is difficult to audit the training process and verify that the model has not been biased or tampered with. This makes it challenging for model builders to comply with any AI regulations that require transparency or auditability of the training process as a precondition for deployment.

An alternative approach for privacy-preserving ML is *confidential computing*.[21] Confidential computing enables the secure execution of code and data in untrusted computing environments—for example, public clouds—by leveraging hardware-based trusted execution environments (TEEs), such as Intel Software Guard Extensions (SGX),[2,5] AMD Secure Encrypted Virtualization-Secure Nested Paging (SEV-SNP),[1] Arm Confidential Compute Architecture (CCA),[15] and more recently, Nvidia Hopper Confidential Computing.[6]

Confidential computing protects the confidentiality and integrity of ML models and data throughout their lifecycles, even from privileged attackers. However, in most existing ML systems with confidential computing, the training process remains centralized, requiring data owners to send (potentially encrypted) datasets to a single client where the model is trained in a TEE. Unlike FL, this setup places significant trust in the TEE infrastructure to protect datasets in a remote, potentially hostile environment.

FL and confidential computing should not be considered competing technologies. Rather, it is possible, with careful design, to combine FL and confidential computing to achieve the best of both worlds: the assurance of sensitive data remaining within its trust domain while ensuring transparency and accountability. This new paradigm, referred to here as *confidential federated learning (CFL)*, can prevent large classes of attacks on FL, broaden the adoption of FL in privacy-sensitive domains, and enable compliance with upcoming AI regulations.

Confidential Computing

Confidential computing uses TEEs to isolate sensitive code and data from privileged attackers. There are several kinds of TEEs in modern CPUs. For example, Intel CPUs support the creation of process-based TEEs through Software Guard Extensions.[2] Process-based TEEs can measure and isolate a user-space process from the rest of the system, including other processes and the operating system (OS). Within process-based TEEs, code does not have direct access to any OS kernel functionality such as I/O devices. Therefore, writing applications to use process-based TEEs requires significant developer effort.

Led by AMD SEV-SNP,[1] recent CPUs support virtual machine (VM)-based TEEs, which can host and isolate both user-mode processes and a full OS from external access. This makes it simpler to migrate existing applications to VM-based TEEs, albeit at the cost of a larger TCB.

While confidential computing has been supported in CPUs for well over a decade, the primitives required for deploying AI workloads such as FL transparently with low performance overheads have evolved only recently.

Confidential containers. While VM-based TEEs can host legacy virtual machines, this mode of deployment has limitations beyond a large TCB. Unless configured correctly, it does not fully isolate the workload (user-mode applications) from external access (for example, secure shell access by the OS admin). It also provides limited attestation of the workload because it requires the VM to be started with a bootloader, which in turn boots an OS kernel. Therefore, only the bootloader is measured by the hardware. Even if attestation were to be extended to include the OS kernel (for example, using a virtual Trusted Platform Module), it is challenging to attest the entire OS and user-mode applications.

Confidential containers[3,11] present a new mode of deploying applications in VM-based TEEs that address these limitations. In confidential containers, a VM-based TEE is used to host a utility OS along with a container runtime, which in turn can host containerized workloads. Confidential containers support full workload integrity and attestation via container execution policies. These policies define the set of container images (represented by the hash digest of each image layer) that can be hosted in the TEE, along with other security-critical attributes, such as commands, privileges, and environment variables. The policy itself is measured (as an initialization time claim) by the hardware root of trust, included in the hardware-signed attestation report and enforced by the container runtime. In

other words, the combination of the OS, container runtime, and container policy fully represents the workload hosted in the TEE and can be used by relying parties to establish trust in the environment.

Confidential GPUs. Initially, support for confidential computing was limited to CPUs, with all other devices considered as untrusted. This was, of course, limiting for AI applications that use GPUs to achieve high performance. Over the past few years, several attempts have been made at building confidential computing support in accelerators. NVIDIA's Hopper generation of GPUs[6] supports the creation of TEEs and can be coupled with CPU-based TEEs (AMD SEV-SNP, Intel TDX[4]) to create a unified TEE across CPU and GPU, enabling transparent offload with low performance overheads.

Hopper GPUs support the new confidential computing mode in which the GPU carves out a region of memory called the protected region and enables a hardware firewall that isolates this region and other sensitive parts of state from the host CPU. In this mode, a CPU-based TEE, such as an SNP VM, can attest and establish a secure channel with the GPU and provision encryption keys to copy engines in the GPU. All subsequent data transfers—including code; models; and application data between the CPU TEE and the GPU, and between GPUs—are encrypted using these keys.

Confidential Federated Learning

A typical FL deployment involves several components that work together to enable collaborative model training across multiple clients. This includes client environments that hold local data, a central aggregator, an orchestrator for managing FL tasks, and the communication infrastructure for provisioning tasks and exchanging model updates.

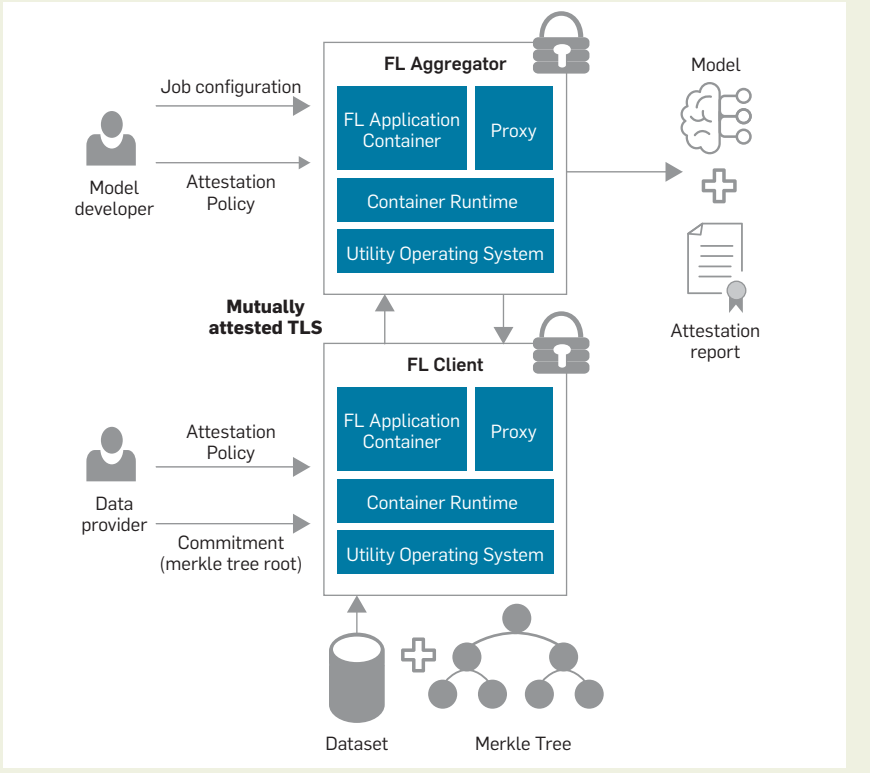
Most FL frameworks, such as NVFlare,[20] support several security measures to protect data and models, including the use of network security to isolate and sandbox remote code; transport layer security (TLS) for secure communication; and strong authentication and access-control mechanisms. Despite these measures, there are plenty of avenues for a malicious participant to exfiltrate secrets or tamper with the training process. For example, a malicious participant can poison datasets by adding samples or changing labels of training data to introduce back doors or bias into the model. Data may be poisoned either before a training job or adaptively during the job, based on intermediate models. A participant may also observe or tamper with gradient updates or arbitrarily tamper with the workflow—for example, by skipping training entirely or not aggregating certain inputs.

CFL is an emerging paradigm[18,19] that aims to harden FL deployments against such attacks. Figure 2 shows the architecture of a typical CFL deployment for a single training job. In CFL, all computation (aggregation and training) is hosted in a special class of hardware-isolated TEEs, which isolate data and computation from all external access, including administrators and privileged attackers. With TEEs in particular, model weights are no longer exposed to client administrators; they are visible only to attested client code. Similarly, intermediate gradient updates are no longer exposed to the aggregator; they are exposed only to attested aggregator code. The aggregator learns the trained model only, and even that access can be limited by hosting the trained model in a TEE.

TEEs used in CFL also provide integrity—a malicious aggregator or client cannot tamper with data, computation, or configuration of the deployment. For example, if a training job requires each client to pre-process the dataset (for example, run sampling and reweighing with specific parameters to mitigate bias),[13] clients cannot change the control flow of the training job or parameter values without being detected via attestation. The integrity properties of TEEs hold even in the presence of side-channel attacks.[12,14,16,23]

Finally, CFL uses TEEs that can provide hardware-based attestation for the full workload and configuration of the FL job, including pre-processing, training, and optional inferencing. TEEs that meet these requirements include Azure Confidential Containers and Confidential Spaces on the Google Cloud Platform.

Figure 2. Architecture of a typical CFL deployment.



Commitments. In addition to hosting computations in TEEs, CFL can support transparency and accountability through *commitments*. Participants in CFL can be required to commit to their inputs before running a training job. Data providers commit to their datasets, and model providers commit to the job configuration and the initial model state (if provided externally). For example, the job configuration in NVFlare is a list of tasks that will be executed by the aggregator and clients, along with the configuration for each task.

Commitments can take various forms. For smaller inputs, such as a job configuration, the input (or its hash digest) can be attested directly. For larger inputs, such as datasets, one option is to compute a Merkle hash tree over the dataset (for example, using dm-verity) and use the root hash of the tree (combined with a random nonce) as a commitment.

In CFL, commitments are reflected in TEE attestation, verified by other participants, and enforced during TEE execution. For example, in an implementation with Azure Confidential Containers, the dm-verity root hash of the training dataset is included as an environment variable in the container security policy. Within the TEE, this root hash is used to verify that the Merkle tree is correct. The Merkle tree is then used to verify the integrity of the dataset by comparing the hash digest of each block that is read against the hash value in the Merkle tree. Reflecting commitments in attestation ensures that any given client can connect to the aggregator only if it provides the committed dataset as input. This invariant holds even across clients and aggregator restarts, since clients and aggregators mutually attest each other on every connection.

Commitments, as used in CFL, have a few noteworthy characteristics. First, they do not impact privacy since only a hash is revealed, not the dataset itself. Commitments do not prevent clients from providing bad data; they ensure only that a malicious client cannot change the dataset adaptively during training. This significantly limits the power of an attacker because the attack must now be designed to work irrespective of other datasets used in training. Finally, commitments, in conjunction with attestation reports, provide tamper-proof provenance for the entire FL job. Armed with attestation reports, external auditors get full visibility into the flow of datasets that contributed to the model and can hold participants responsible for a model's behavior.

Confidential computing enables the secure execution of code and data in untrusted computing environments by leveraging hardware-based trusted execution environments.

Mutual attestation. Including the full workload, configuration, and commitments in attestation reports enables other participants in an FL computation to remotely verify and establish trust in a participant's compute instances. For example, an aggregator can verify all clients, and each client can independently verify the central aggregator.

In CFL, each participant specifies its criteria for trusting other participants by creating an attestation policy. This can take the form of a key-value map, where each key is the name of a claim, and the value is the set of values that the claim is allowed to take.

The following is a sample attestation policy with multiple claims and permitted values for each claim. Each CFL node is provisioned with a policy that it uses to verify attestation reports from other nodes.

```
{
 "host_data": ["..."],
 "report_data": ["...", "...", "...", ]
 "svn": [ "..." ]
}
```

To ensure that a participant communicates only with other participants that it trusts, CFL deployments can perform attestation verification as part of the TLS handshake:

1. On start-up, each client and aggregator generates an ephemeral TLS signing key and obtains an attestation report with the key as a runtime claim.

2. Each node generates a self-signed certificate and includes the attestation report and other collateral required to verify the report (such as device certificates) as a custom extension in the certificate. Each instance configures its TLS stack to use this TLS signing certificate.

3. Each node also configures the TLS stack (for example, using call-

backs supported by TLS) to verify certificates obtained from other participants during the handshake, based on its attestation policy. This protocol ensures that each instance establishes a secure encrypted communication channel with other instances only after verifying the attestation report against the attestation policy. All subsequent communication between the aggregator and client, such as communicating model weights and gradient updates, uses this channel.

One challenge in deploying attestation policies is that it can lead to cyclic dependencies, because the aggregator's attestation policy depends on each client's attestation, and vice versa. One way to break the cycle is to include the aggregator's attestation policy in its attestation but exclude the client's policy from its attestation. This design choice preserves the ability for clients to assess the aggregator's attestation policy before entrusting the aggregator with their data.

Implementing CFL

We have experimented with a CFL implementation based on NVIDIA NVFlare, a commonly used FL framework. Our prototype can run on confidential containers on Azure Container Instances (ACIs) as well as confidential VMs (CVMs).[9] NVFlare containers could be hosted in ACI and CVMs without modifying the core NVFlare framework. To simplify deployment, we built a provisioning tool to generate scripts for generating dataset commitments, attestation policies for clients and servers, and scripts for deploying NVFlare containers to ACIs and CVMs. Dataset commitments are implemented using dm-verity. Transparent, mutually attested TLS and attestation policy enforcement are supported using a network proxy.

We evaluated the CFL's end-to-end performance by measuring the training throughput. To perform the end-to-end evaluation, we deploy the CFL aggregator in Azure DC4asv5 CVM (with four vCPUs, 16GB of memory) and the CFL client in Azure DC32asv5 CVMs (32 vCPUs, 128GB of memory). Our experiments suggest that adding TEE and dm-verity protection for the FL system results in a 5% reduction in overall throughput for simple FL workloads (based on CIFAR-10).

We also investigated the overhead of introducing commitments using dm-verity with a sequential read benchmark, which is representative of training workloads where the entire dataset is read sequentially. Our experiments suggest dm-verity protection can introduce an overhead up to 40% in sequential read throughput as a result of read amplification caused by Merkle tree checks. The impact of reduced storage throughput on end-to-end training throughput is small because most training workloads tend to be compute-bound. These are initial results and need to be substantiated with more rigorous evaluation using larger workloads.

Conclusion

The principles of security, privacy, accountability, transparency, and fairness are the cornerstones of modern AI regulations. Classic FL was designed with a strong emphasis on security and privacy at the cost of transparency and accountability. Confidential federated learning addresses this gap with a careful combination of FL with TEEs and commitments. CFL also brings other desirable security properties, such as code-based access control, model confidentiality, and protection of models during inference. Recent advances in confidential computing, such as confidential containers and confidential GPUs, mean that existing FL frameworks can be extended seamlessly to support CFL with low overheads. For these reasons, CFL is likely to become the default mode for deploying FL workloads. C

References

1. AMD. AMD SEV-SNP: Strengthening VM isolation with integrity protection and more. White Paper (2020); https://bit.ly/3zE4vec.
2. Anati, I., Gueron, S., Johnson, S., and Scarlata, V. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd Intern. Workshop on Hardware and Architectural Support for Security and Privacy* (2013); https://intel.ly/3S1kpFC.
3. Brasser, F. et al. Trusted container extensions for container-based confidential computing. *arXiv* (2022): https://arxiv.org/abs/2205.05747.
4. Cheng, P.-C. et al. Intel TDX demystified: A top-down approach. *arXiv* (2023); https://arxiv.org/abs/2303.15540.
5. Costan, V. and Devadas, S.. Intel SGX explained (*2016*); https://eprint.iacr.org/2016/086.
6. Dhanuskodi, G. et al. Creating the first confidential GPUs. *ACM Queue 21*, 4 (2023); https://queue.acm.org/detail.cfm?id=3623391.
7. Dwork, C. et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science 9*, 3–4 (2014), 211–407; 10.1561/0400000042.
8. Fang, M., Cao, X., Jia, J., and Gong, N. Local model poisoning attacks to Byzantine-robust federated learning. In *Proceedings of the 29th Usenix Security Symp., article 92* (2020), 1623–1640; https://bit.ly/4f2t84z.
9. Hande, K. Announcing Azure confidential VMs with NVIDIA H100 Tensor Core GPUs in preview. *Azure Confidential Computing Blog* (Nov. 15, 2023); https://bit.ly/3VXtnFf
10. Jere, M.S., Farnan, T., and Koushanfar, F.. A taxonomy of attacks on federated learning. In *Proceedings of IEEE Security & Privacy 19*, 2 (2020), 20–28; https://ieeexplore.ieee.org/document/9308910.
11. Johnson, M.A. et al. COCOAEXPO: Confidential containers via attested execution policies. *arXiv* (2023); https://arxiv.org/abs/2302.03976.
12. Kocher, P. et al. Spectre attacks: exploiting speculative execution. In *Proceedings of the 40th IEEE Symp. on Security and Privacy* (2019), 1–19; https://ieeexplore.ieee.org/document/8835233.
13. Krasanakis, E., Spyromitros-Xioufis, E., Papadopoulos, S., and Kompatsiaris, Y. Adaptive sensitive reweighting to mitigate bias in fairness-aware classification. In *Proceedings of the 2018 World Wide Web Conf.*, 853–862; 10.1145/3178876.3186133.
14. Li, M. et al. CIPHERLEAKS: Breaking constant-time cryptography on AMD SEV via the ciphertext side channel. In *Proceedings of the 30th Usenix Security Symp.* (2021), 717–732.
15. Li, X. et al. Design and verification of the Arm confidential compute architecture. In *Proceedings of the 16th Usenix Symp. on Operating Systems Design and Implementation* (2022); https://bit.ly/3zGvpSH
16. Lipp, M. et al. Meltdown: reading kernel memory from user space. In *Proceedings of the 27th Usenix Security Symp.*; https://bit.ly/45YQzr6.
17. McMahan, B. et al. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th Intern. Conf. on Artificial Intelligence and Statistics* (2017), 1273–1282; https://bit.ly/3XUaHZD
18. Mo, F. et al. PPFL: Enhancing privacy in federated learning with confidential computing. *GetMobile: Mobile Computing and Communications 25*, 4 (2022), 35–38; https://bit.ly/3xGWFQw.
19. Quoc, D.L. and Fetzer, C. SecFL: confidential federated learning using TEEs. *arXiv 2110.00981* (2021); https://arxiv.org/abs/2110.00981.
20. Roth, H.R. et al. NVIDIA Flare: federated learning from simulation to real-world. *arXiv* (2022); https://arxiv.org/abs/2210.13291.
21. Russinovich, M. et al. Toward confidential cloud computing. *Communications of the ACM 64*, 6 (2021), 54–61; 10.1145/3453930.
22. Tolpegin, V., Truex, S., Gursoy, M.E., and Liu, L. Data poisoning attacks against federated learning systems. In *Proceedings of the 25th European Symp. on Research in Computer Security, Part I 25* (2020), 480–501; https://bit.ly/3WgHaIq.
23. Van Bulck, J. et al Foreshadow: extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th Usenix Security Symp.* (2018); https://bit.ly/3LlwdPu
24. Wei, K. et al. Federated learning with differential privacy: algorithms and performance analysis. In *Proceedings of IEEE Transactions on Information Forensics and Security 15* (2020), 3454–3469; https://bit.ly/3VSmlS8

Jinnan Guo is a Ph.D. candidate at Imperial College London, U.K., advised by Peter Pietzuch. His research interest lies in the intersection of systems, security, and machine learning.

Peter Pietzuch is a professor of distributed systems at Imperial College London, U.K., where he leads the Large-scale Data & Systems (LSDS) group.

Andrew Paverd is a principal research manager in the Microsoft Security Response Center (MSRC). His research work focuses primarily on security, privacy, and safety in AI systems.

Kapil Vaswani is a principal researcher at Azure Research in Cambridge, England, U.K. His research interests lie in secure and robust systems.