Private delegated computations using strong isolation

Mathias Brossard[†], Guilhem Bryant[†], Basma El Gaabouri[†], Xinxin Fan[⋆], Alexandre Ferreira[†]
Edmund Grimley-Evans[†], Christopher Haster[†], Evan Johnson[‡], Derek Miller[†], Fan Mo[¶]
Dominic P. Mulligan[†], Nick Spinale[†], Eric van Hensbergen[†], Hugo J. M. Vincent[†], Shale Xiong[†]

[†]*Systems Group, Arm Research* [⋆]*IoTeX.io*
[‡]*University of California, San Diego* [¶]*Imperial College London*

Abstract

Sensitive computations are now routinely *delegated* to third-parties. In response, *Confidential Computing* technologies are being introduced to microprocessors, offering a protected processing environment, which we generically call an *isolate*, providing confidentiality and integrity guarantees to code and data hosted within—even in the face of a privileged attacker. Isolates, with an attestation protocol, permit remote third-parties to establish a trusted "beachhead" containing known code and data on an otherwise untrusted machine. Yet, the rise of these technologies introduces many new problems, including: how to ease provisioning of computations safely into isolates; how to develop distributed systems spanning multiple classes of isolate; and what to do about the billions of "legacy" devices without support for Confidential Computing?

Tackling the problems above, we introduce *Veracruz*, a framework that eases the design and implementation of complex privacy-preserving, collaborative, delegated computations among a group of mutually mistrusting principals. Veracruz supports multiple isolation technologies and provides a common programming model and attestation protocol across all of them, smoothing deployment of delegated computations over supported technologies. We demonstrate Veracruz in operation, on private in-cloud object detection on encrypted video streaming from a video camera. In addition to supporting hardware-backed isolates—like AWS Nitro Enclaves and Arm® Confidential Computing Architecture Realms—Veracruz also provides pragmatic "software isolates" on Armv8-A devices without hardware Confidential Computing capability, using the high-assurance *seL4* microkernel and our *IceCap* framework.

1 Introduction

Code and data are now routinely shared with a *delegate* who is better placed, either through economies of scale, or computational capacity, to host a computation. While Cloud computing is the obvious exemplar of this trend, other forms of distributed computing—including volunteer Grid Computing, wherein machines lend spare computational capacity to realize some large computation, and Ambient Computing, wherein computations are *mobile* and hop from device-to-device as computational contexts change—also see computations freely delegated to third parties.

At present, in the absence of the widespread deployment of Advanced Cryptography [42], delegating computation to a third party inexorably means entering into a trust relationship with the delegate, and for some especially sensitive computations this may be simply unacceptable. Yet, even for less sensitive delegated computations, there is still an interest in limiting the scope of this trust relationship. In the Cloud context, though established hosts may be reputable, technical means may be desired to shield computations from prying or interference which may originate from many sources, not only from the hosting company themselves: malefactors may exploit hypervisor bugs to spy on co-tenants, for example. Cloud hosts also increasingly see an interest in *deniable hosting*, wherein technical measures ensure that a customer's computations simply cannot be interfered with, or spied upon, by the hosts themselves—even in the face of legal compulsion. For Ambient and volunteer Grid Computing, these concerns also manifest: nodes must be assumed hostile and assumed to be trying to *undermine* a computation, either through malice or as a consequence of bugs or glitches. As a result, volunteer Grid Computing deployments may schedule computations on multiple nodes and check for consistency [4].

In response, novel *Confidential Computing* technologies are being added to microprocessor architectures and cloud infrastructure, providing protected computing environments—variously called Secure Enclaves, Realms, Trusted Execution Environments, and which we generically call *isolates*—that provide strong confidentiality and integrity guarantees to code and data hosted within, even in the face of a privileged attacker. Isolates are also typically paired with an *attestation* protocol, allowing a third-party to deduce, with high confi-

dence, that a remote isolate is authentic and configured in a particular way. Taken together, one may establish a protected "beachhead" on an untrusted third-party's machine—exactly what is needed to protect delegated computations.

Isolates offer a range of benefits for system designers, namely allowing programmers to design arbitrarily complex privacy-preserving distributed systems using standard tools and programming idioms that run at close to native speed. Moreover, compared to cryptographic alternatives, Confidential Computing technology is available for use and deployment in real systems *today*. Yet, the emergence of Confidential Computing technology poses some interesting problems.

First, note that Confidential Computing technologies simply provide an empty, albeit secure, isolate. Associated questions like how computations are securely provisioned into an isolate, how to make this process straightforward and foolproof, and how systems are designed and built around isolates as a new kind of primitive, are left unanswered. Moreover, for some types of distributed system—such as Grid and Ambient computing systems, previously discussed—it is feasible that *different* types of isolate will be used within a single larger system. Here, bridging differences in attestation protocol and programming model will be key, as will be easing deployment and scheduling of computations hosted within isolates.

For this reason, we introduce our main research contribution: *Veracruz*, a framework that abstracts over isolates and their associated attestation processes. Veracruz supports multiple different isolation technologies, including hardware-backed isolates like AWS Nitro Enclaves and Arm Confidential Computing Architecture Realms on a private branch. Adding support for more is straightforward. Veracruz provides a uniform programming model across different supported isolates—using WebAssembly (Wasm, henceforth) [96]—and a generalized form of attestation, providing a "write once, isolate anywhere" style of development: programs can be protected using *any* supported isolation technology without recompilation. Veracruz is discussed in §4.

Veracruz captures a particularly general form of interaction between mutually mistrusting parties. As a result, Veracruz can be specialized in a straightforward manner to obtain an array of delegated, privacy-preserving computations of interest. In support of this claim we provide a description of how Veracruz can be used for secure ML model aggregation, and an industrial case-study built around AWS Nitro Enclaves, demonstrating an end-to-end encrypted video decoding and object-detection flow, using a deep learning framework processing video obtained from an IoT camera. These case-studies, and further benchmarking, are discussed in §5.

In §2 we argue that Confidential Computing technology is likely to be widely deployed within industry, despite well-known flaws in particular implementations. Yet, *billions* of existing devices have already been shipped without any explicit support for Confidential Computing, and these devices will continue to be used for years, if not decades, to come. Is there some *pragmatic* isolation mechanism that we could use on "legacy" devices which, while falling short of the confidentiality and integrity guarantees offered by hardware-backed Confidential Computing mechanisms, can yet provide believable isolation for workloads? Rising to this challenge, we introduce our second research contribution: *IceCap*, a pragmatic "software isolate" for Armv8-A devices without explicit support for Confidential Computing. IceCap uses the high-assurance *seL4* microkernel to provide strong confidentiality and integrity guarantees for VMs, with little overhead.

IceCap is supported by Veracruz, and taken together, one may design and deploy delegated computations across hardware- and software-isolates on next-generation and legacy hardware, alike. We introduce IceCap in §3, as a stepping stone to the introduction of Veracruz.

2 Hardware-backed Confidential Computing

In addition to the already widely-deployed *Arm TrustZone®* [7] and *Intel Software Guard Extensions* (SGX) [29], an emerging group of novel *Confidential Computing* technologies are being added to microprocessor architectures and cloud infrastructures, including *AMD Secure Encrypted Virtualization* (SEV) [50], *Arm Confidential Computing Architecture* (CCA) [6], *AWS Nitro Enclaves* [9], and *Intel Trust Domain Extensions* (TDX) [46]. All introduce a hardware-backed protected execution environment, which we call an **isolate**, providing strong **confidentiality** (the content of the isolate remains opaque to external observers) and **integrity** (the content of the isolate remains protected from interference by external observers) guarantees to code and data hosted within. These guarantees apply even in the face of a strong adversary, with any operating system or, in most cases even a hypervisor, outside of the isolate assumed hostile. Memory encryption may also be provided as a standard feature to protect against a class of physical attack. Isolates are often associated with an **attestation protocol**—e.g., *EPID* for Intel SGX [14, 15] and *AWS Nitro Attestation* for AWS Nitro Enclaves [9]. These permit a third party to garner strong, cryptographic evidence of the authenticity and configuration of a remote isolate.

Some isolate implementations have unfortunately fallen short of their promised confidentiality and integrity guarantees. A substantial body of academic work, demonstrating that side-channel (see e.g. [13, 16, 22, 30, 44, 64, 87, 88, 98, 101]) and fault injection attacks [24, 67, 84] can be used to exfiltrate secrets from isolates, now exists, and a perception—at least in the academic community and technical press—appears to be forming that isolates are fundamentally broken and any consequent research project that builds upon them need necessarily justify that decision. We argue that this

emerging perception is an instance of *the perfect being the enemy of the good*.

First, we expect that many identified flaws will be gradually ironed out over time, either in point-fixes, iterated designs, or by the adoption of software models that avoid known vulnerabilities. For hardware, we have already seen some flaws fixed using microcode updates and other point-fixes by affected manufacturers (e.g, [27]). For software, research into methods designed to avoid known classes of side-channels is emerging, through implementation techniques such as constant-time algorithms, and dedicated type-systems such as FaCT [19] and CT-Wasm [95]. These may prove to be useful in implementing systems with isolates, and we summarize our own ongoing experimentation with these approaches in §6.

Second, we expect that industrial adoption of isolates will be widespread, and arguably this is already in evidence with the formation of consortia such as the LF's *Confidential Computing Consortium* [25], and an emerging ecosystem of industrial users and startups. Researching systems that use isolates, and ease their deployment, is therefore not only justifiable, but very useful. Here, industrial users pragmatically evaluate isolate-based systems in comparison with the *status quo*, where delegated computations are—by and large—left completely unprotected, and we argue that it is this standard which should be applied when evaluating systems built around isolates, not comparison with side-channel free cryptography which is still impractical in an industrial context. In this light, forcing malefactors to resort to side-channel and fault injection attacks—many of which are impractical, or can be defended against using others means—to exfiltrate data from an isolate is a welcome, albeit incremental, improvement in the privacy-guarantees that real systems can offer users.

3 IceCap

IceCap is a hypervisor with a minimal trusted computing base (TCB, henceforth) built around the formally verified *seL4* microkernel. IceCap provides a pragmatic and flexible *software isolate* for many existing Armv8-A devices. The IceCap hypervisor relegates the untrusted operator to a domain of limited privilege called the host. This domain consists of a distinguished virtual machine—housing a rich operating system such as Linux—and a minimal accompanying virtual machine monitor. The host domain manages the device's CPU and memory resources, and drives device peripherals which the TCB does not depend on. This includes opaque memory and CPU resources for confidential virtual machines—or isolates. However, the host does not have the right to access the resources of isolates—while scheduling and memory management *policy* is controlled by the host, *mechanism* is the responsibility of more trustworthy components.

IceCap's TCB includes the seL4 microkernel and compartmentalized, privileged seL4-native services running in EL0. These co-operate defensively with the host to expose isolate lifecycle, scheduling, and memory management mechanisms.

At system initialization, the hypervisor extends from the device's root of trust via a device-specific measured boot process and then passes control to the untrusted host domain. A remote party coordinates with the host to spawn a new isolate by first sending a declarative specification of the isolate's initial state to IceCap's *trusted spawning service*, via the host, which then carves-out the requested memory and CPU resources from resources which are inaccessible to the host. A process on the host, called the *shadow virtual machine monitor*, provides untrusted paravirtualized device backends to isolates, and also acts as a *token* representing the isolate in the host's scheduler, to enable the host operating system to manage isolate scheduling policy with minimal modification.

To support attestation of isolates, IceCap would use a platform-specific measured boot to prove its own identity and then attest that of an isolate to a remote challenger. This is not yet implemented, with IceCap attestation being stubbed to support Veracruz, but straightforward to do so.

seL4 is accompanied by security and functional correctness proofs, checked in *Isabelle/HOL* [68, 69, 82], providing assurance that IceCap correctly protects isolates from software attacks. By using seL4, IceCap will also benefit from ongoing research into the elimination of certain classes of timing channels [38]. The trusted seL4 userspace components of IceCap are not yet verified, though they are compartmentalized and initialized using *CapDL* [55], which has a formal semantics known to be amenable to verification [18] from previous work. Using the high-level seL4 API, these components are also implemented at a high level of abstraction in Rust, making auditing easier and eliminating the need to subvert the Rust compiler's memory safety checks—even for components which interact with hardware address translation structures. The IceCap TCB is small and limited in scope—about 40,000 lines of code. Virtual machine monitors are moved to the trust domains of the virtual machines they supervise, thereby eliminating emulation code from the TCB. Towards that end, cross-domain fault handling is replaced with higher-level message passing via seL4 IPC.

Isolates are also protected with the System MMU (SMMU) from attacks originating from peripherals under the host's control. IceCap is designed to seamlessly take advantage of additional hardware security features based on, or aligned with, address translation-based access controls—Arm TrustZone [7], for example. TrustZone firmware typically uses the `NS` state bit to implement a coarse context switch, logically partitioning execution on the application processor into two *worlds*. IceCap could use this to run isolates out of secure-world memory resources, protected by platform-specific mechanisms which may mitigate certain

	Events per second (via `sysbench`)	
	Host	Guest
Firecracker	586.18	582.65 (-0.60%)
IceCap	583.68 (-0.43%)	572.28 (-2.18%)
	Bandwidth (Gbits/sec)	
	Guest → Host	Host → Guest
Firecracker	3.42	3.14
IceCap	3.08 (-9.9%)	3.18 (+1.3%)

Table 1: Overheads for IceCap compute-bound workloads (top) and virtual network performance (bottom)

classes of physical attack.

Under IceCap, isolate and host incur a minimal performance overhead compared to host and guests under *KVM* [51]. We use *Firecracker* [2]—an open-source VMM for KVM from AWS—as a point of comparison, due to its minimalism for the sake of performance, and preference for paravirtualization over emulation. Compute-bound workloads in IceCap isolates incur a ∼2.2% overhead compared to native Linux processes and a ∼1.8% overhead compared to Firecracker guests due to context switches through the TCB on timer ticks (see Table 1). The virtual network bandwidth between the host and an isolate represents how data flows through IceCap in bulk. However, at the time of writing, untrusted network device emulation differs from Firecracker's trusted network device emulation in ways that hinder a satisfying comparison, and with this in mind, we note guest-to-host incurs a ∼9.9% bandwidth overhead, whereas host-to-guest outperforms Firecracker by a small margin. As IceCap's implementation matures, we expect virtual network bandwidth overhead to settle between these two points.

The great performance of seL4 IPC [81] helps reduce IceCap's performance overhead, and this is further helped by minimizing VM exits using aggressive paravirtualization: VMMs for both host and guest do not even map any of their VMs' memory into their own address spaces, and their only runtime responsibility is emulating the interrupt controller, with their VMs employing interrupt mitigation to even avoid that.

Next, we introduce a framework for designing and deploying privacy-preserving delegated computations across various different isolation technologies—IceCap included.

4 Veracruz

Throughout this section we make reference to the system components presented in the schematic in Fig. 1.

Veracruz is a *framework* which may be specialized to obtain a particular privacy-preserving, collaborative computation of interest. A Veracruz computation involves an arbitrary number of **data owners**, trying to collaborate with a single **program owner**. The framework places no limits on the number of data owners, but a particular computation obtained by specializing Veracruz will always spell out a precise number of participants. We use π to denote the program of the program owner, and use D_i for $1 \leq i \leq N$ to denote the data sets of the various data owners in an arbitrary Veracruz computation.

Collectively, the goal of the various principals, Ⓟ, is straightforward: they wish to compute the value $\pi(D_1, \ldots, D_N)$, that is, the value of the program π applied to the N inputs of the various data owners. To do this, they may choose to make use of a third party machine to power the computation, Ⓓ. We refer to the owner of this machine as the **delegate**, and this machine is assumed capable of launching an isolate of a type that Veracruz supports, loaded with the Veracruz **trusted runtime**, Ⓥ. This runtime acts as a "neutral ground" within which a computation takes place, and provides strong **sandboxing** guarantees to the delegate, who is loading untrusted code in the form of π, onto their machine. The runtime is open-source, and auditable by principals, assuming bit-for-bit reproducible builds.

Each principal in a Veracruz computation has a mixture of **roles**, consisting of some combination of **data provider**, **program provider**, **delegate**, and **result receiver**. While the first three have been implicitly introduced, the latter role refers to principals who will receive the result of the computation. The identification details of each principal, in the form of cryptographic certificates (or an IP address for the delegate), and their mixture of roles, is captured in a public **global policy** configuration file, Ⓛ, which parameterizes each computation, and which also contains other important bits of metadata. Only one principal may be delegate or program provider.

The global policy captures the **topology** of a computation, specifying where information may flow from, and to whom, in a computation, while varying the program π varies precisely **what** is being computed. By varying the two, Veracruz can capture a general pattern of interaction shared by many delegated computations, and one could, for example, effect a varied palette of computations of interest, including:

Moving heavy computations safely off a computationally-weak device to an untrusted edge device or server. The computationally-weak device is both data provider and result receiver, the untrusted edge device or server is delegate, and the computationally-weak device or its owner is the program provider, providing the computation to be performed.

Privacy-preserving machine learning between a pair of mutually distrusting parties with private datasets, but where learnt models are made available to both participants. Both principals are data providers, contributing their datasets provided in some common format, and also act as result receivers for the learnt model. Arbitrarily one acts as program provider, providing the implementation of the machine learning algorithm of interest. A third-party, e.g., a Cloud host,

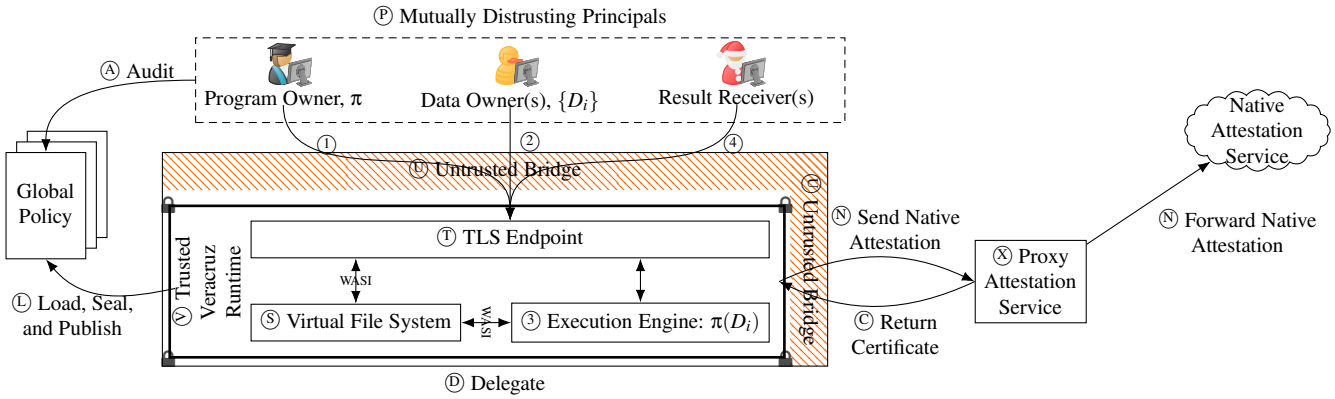

Figure 1: An overview of an abstract Veracruz computation, showing principals and their roles, major system components, and a suggestive depiction of data-flow. Isolates, such as those hosting the Veracruz runtime, are marked with boxes with padlocks

acts as delegate.

A DRM mechanism wherein novel IP (e.g., computer vision algorithms) are licensed out on a "per use" basis, and where the IP is never exposed to customers. The IP owner is program provider, and the licensee is both data provider and result receiver, providing the inputs to, and receiving the output from, the private IP. The IP owner themselves may act as delegate, or this can be contracted out to a third-party. With this, the IP owner never observes the input or output of the computation, and the licensee never observes the IP.

The implementation of privacy-preserving auctions. An auction service acts as program provider, implementing a sealed-bid auction, and also acts as delegate. Bidders are data providers, submitting sealed bids. All principals are also result receivers, receiving notice of the auction winner and the price to be paid, which is public. Neither bidder nor auction service ever learn the details of any bids, other than their own and the winning bid.

In addition, it is easy to see how more complex distributed systems can be built *around* Veracruz. For example, a volunteer Grid computing framework where confidentiality is not paramount, but computational integrity is; an Ambient computing runtime for mobile computations across a range of devices; a privacy-preserving MapReduce [32] or Function-as-a-Service (FaaS, henceforth) style framework. Here, computational nodes act as an independent delegate for some aspect of the wider computation, and *different* isolation technologies may also be used in a single computation, either due to availability for Grid or Ambient computing, or due to scheduling of sensitive sub-computations onto stronger isolation mechanisms for MapReduce.

In the most general case, each principal in a Veracruz computation is mutually mistrusting, and does not wish to **declassify**—or intentionally reveal—their data: data providers do not wish to divulge their input datasets and the program provider does not wish to divulge their program. Nevertheless, as the examples enumerated above indicate, for some computations declassification *can* be useful, for example as inducement to other principals to enroll in the computation, a "nothing up my sleeve" demonstration. Referring back to the privacy-preserving machine learning use-case, above, the program provider may *intentionally* declassify their program for auditing—before other principals agree to participate—as a demonstration that the program implements the correct algorithm, and will not (un)intentionally leak secrets. Similarly, for a Grid computing project, revealing details of the computation, as an enticement to users to donate their spare computational capacity, may be beneficial.

Declassification can also occur as a side effect of the computation itself, for example when the result of a computation—which can reveal significant amounts of information about its inputs, depending on π—is shared with an untrusted principal. Principals must evaluate the global policy carefully, before enrolling, to understand where results will flow to, and what they may say about any secrets. Though Veracruz can be used to design privacy-preserving distributed computations, not *every* computation is necessarily privacy-preserving.

Once the delegate has spawned an isolate with the Veracruz runtime loaded, the program and data owners establish a TLS connection, using a modified TLS handshake, with the isolate Ⓣ, as will be described later in §4.1. This handshake assures the principals that the isolate is, in fact, executing the Veracruz runtime specified in the global policy, and that the isolate is the other end of their TLS connection. Once this TLS channel is established, the program and data providers use it to **provision** their respective secrets directly into the isolate, ① and ②. This makes use of an untrusted bridge, Ⓤ, on the delegate's machine but outside of the isolate, to forward encrypted TLS data received into the isolate itself. To the delegate, communication via this bridge is immutable and opaque—except for sizing and timing information that TLS leaks—unless they can subvert TLS. Note that TLS configuration options, including permitted ciphersuites, and the `SHA-256` hash of the program π, are also specified in the global policy. This latter aspect ensures that when a program,

π, is declassified, it can be audited by other principals, and verified to be the same program provisioned into the isolate.

Provisioned secrets are stored as files in a virtual, **in-memory filesystem** maintained by the Veracruz runtime, Ⓢ. The contents of this filesystem never leave the isolate, and are destroyed when the isolate is torn down. The paths of data inputs, D_i, are specified in the global policy file, as the program π needs to know where its inputs are stored for processing when the computation starts executing. Similarly, the program π is also stored as a file, and will be read from the filesystem itself when loaded for execution by the runtime.

Once everything is in place, a result receiver may request the result of the computation, triggering the Veracruz runtime to load the provisioned program, π, into the execution engine, Ⓢ, and either compute the result $\pi(D_1, \ldots, D_N)$, terminate with an error code, or diverge. Assuming a result is computed, it is stored by the program as a file in the filesystem at a path specified by the global policy. The runtime reads this path, or fails with an error if the program did not write a result there, and makes the result retrievable securely, via TLS, to all result receivers, ④. The computation is now complete.

4.1 Attestation

Given Veracruz supports multiple isolation technologies, this poses a series of attestation-related problems:

Complex client code: Software used by principals delegating a computation to Veracruz must support multiple attestation protocols, complicating it. As Veracruz adds support for more isolation mechanisms—potentially with new attestation protocols—this client code must be updated to interact with the new class of isolate.

Leaky abstraction: Veracruz abstracts over isolation technology, allowing principals to easily delegate computations without worrying about the programming or attestation model associated with any one class of isolate. Forcing clients to switch attestation protocols, depending on the isolation technology, breaks this uniformity.

Potential side-channel: For some attestation protocols, each principal in a Veracruz computation must refer attestation evidence to an external attestation service.

Attestation policy: principals may wish to disallow computations on delegates with particular isolation technologies. These policies may stem from security disclosures—vulnerabilities in particular firmware versions, for example—changes in business relationships, or geopolitical trends. Given our support for heterogeneous isolation technologies, being able to declaratively specify who or what can be trusted becomes desirable. Existing attestation services do not take policy into account, pushing the burden onto client code—problematic if policy changes, as client code must be updated.

In response, we introduce a **proxy attestation service** for Veracruz, which must be explicitly trusted by all principals to a computation, with associated server and management software open source, and auditable by anyone. This service is not protected by an isolate, though in principle could be, and doing so would allow principals to check the authenticity of the proxy attestation service, before trusting it, for example. Implementing this would be straightforward; for now we assume that the attestation service is trusted, implicitly.

The proxy attestation service first uses an **onboarding** process to enroll an isolate hosting Veracruz, after which the isolate can act as a TLS server for principals participating in a computation. We describe these steps, referring to Fig. 2.

Onboarding an isolate The proxy attestation service maintains a root CA key (a public/private key pair) and a Root CA certificate containing the root CA public key, signed by the root CA private key. This root CA certificate is included in the global policy file of any computation using that proxy attestation service. An onboarding protocol is then followed:

1. Upon initialization inside the isolate, the Veracruz runtime Ⓥ generates an asymmetric key pair, along with a *Certificate Signing Request* (or CSR, henceforth) [71] for that key pair.
2. The Veracruz runtime performs the platform's **native attestation flow** Ⓞ with the proxy attestation server acting as challenger Ⓧ. These native attestation flows provide fields for user-defined data, which we fill with a cryptographic hash (`SHA-256`) of the CSR, which cryptographically binds the CSR to the attestation data, ensuring that they both come from the same isolate. The Veracruz runtime sends the CSR to the proxy attestation server along with the attestation evidence.
3. The proxy attestation server **authenticates** the attestation evidence received via the native attestation flow. Depending on the particular protocol, this could be as simple as verifying signatures via a known-trusted certificate, or by authenticating the received evidence using an external attestation service.
4. The proxy attestation service computes the hash of the received CSR and compares it against the contents of the user-defined field of the attestation evidence. If it matches, it confirms that the CSR is from the same isolate as the evidence.
5. The proxy attestation server converts the CSR to an `X.509` Certificate [28] containing a custom extension capturing details about the isolate derived from the attestation process, including a hash of the Veracruz runtime executing inside the isolate (and optionally other information about the platform on which the isolate is executing). The certificate is signed by the private component of the proxy attestation server's Root CA key.

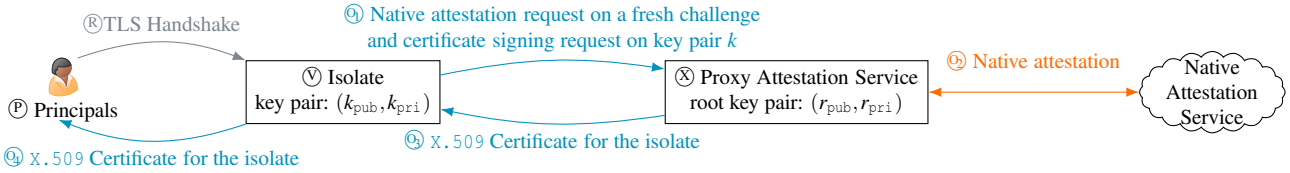

Figure 2: A schematic diagram of the Veracruz attestation service onboarding and challenge protocols

6. The proxy attestation server returns the generated certificate to the Veracruz runtime inside the isolate.

In the typical CA infrastructure, a delegated certificate can be revoked by adding it to a *Certificate Revocation List*, checked by clients before completing a TLS handshake. While this scheme is possible with our system, we elected to use a different approach, setting the expiry in the isolate's certificate to a relatively short time in the future, so that the proxy attestation service can limit the amount of time a compromised isolate can be used in computations. The lifetime of isolate certificates can be decided upon via a policy of the proxy attestation service, based upon their appetite for risk.

Augmented TLS handshake After an isolate is onboarded, Ⓠ, a principal, Ⓡ can attempt to connect to it, using an augmented TLS handshake. In response to the "Client Hello" message sent by the principal, the isolate responds with a "Server Hello" message containing the certificate that the isolate received from the proxy attestation server, described above. The principal then verifies that certificate against the proxy attestation server root CA certificate contained within the global policy. If it matches, it recognizes that the certificate was indeed generated by the proxy attestation server. Recall that this certificate contains a custom extension. Assuming successful verification, the principal then checks the data contained in this extension against the expected values in the global policy. As currently implemented, the extension contains the hash of the Veracruz runtime, which is also listed in the global policy, and the two are checked by the principal. If they match, the principal continues the TLS handshake, confident in the fact that it is talking to a Veracruz runtime executing inside of a supported isolation technology.

Note that the proxy attestation service solves the problems with attestation described above. First, client code is provided with a uniform attestation interface—here, we use Arm's PSA attestation protocol [86]—independent of the underlying isolation technology in use. Second, none of the principals in the computation need to communicate with any native attestation service. Thus, the native attestation service knows that software was started in a supported isolate, but it has no knowledge of the identities or even the number of principals. Finally, the global policy represents the only source of policy enforcement. The authors of the global policy can declaratively describe who and what they are willing to trust, with a principal's client software taking this information into account when authenticating or rejecting an attestation token.

Lastly, we note. that our attestation process is specifically designed to accommodate client code running on embedded microcontrollers—e.g., Arm Cortex®-M3 devices—with limited computational capacity, constrained memory and storage (often measured in tens of kilobytes), and which tend to be battery-powered with limited network capacity. Communication with an attestation service is therefore cost- and power-prohibitive, and using a certificate-based scheme allows constrained devices to authenticate an isolate running Veracruz efficiently. To validate this, we developed Veracruz client code for microcontrollers, using the Zephyr embedded OS [100]. Our client code is 9KB on top of the `mbedtls` stack [60], generally required for secure communication anyway. Using this, small devices can offload large computations safely to an attested Veracruz instance.

4.2 Programming model

Wasm [41] is designed as a sandboxing mechanism for use in security-critical contexts—namely web browsers—designed to be embeddable within a wider host, has a precise semantics [93], is widely supported as a target by a number of high-level programming languages such as Rust and C, and has high-quality interpreters [90] and JIT execution engines available [91]. We have therefore adopted Wasm as our executable format, supporting both interpretation and JIT execution, with the strategy specified in the global policy.

Veracruz uses Wasm to protect the delegate's machine from the executing program, to provide a uniform programming model, to constrain the behavior of the program, and to act as a portable executable format for programs, abstracting away the underlying instruction set architecture. Via Wasm, the trusted Veracruz runtime implements a "two-way isolate" wherein the runtime is protected from prying and interference from the delegate, and the delegate is protected from malicious program behaviors originating from untrusted code.

To complete a computation, a Wasm program needs some way of reading inputs provided to it by the data provider, and some way of writing outputs to the result receivers. However, we would like to constrain the behavior of the program as far as possible: a program dumping one of its secret inputs to `stdout` on the host's machine would break the privacy guarantees that Veracruz aims to provide, for example. Partly for

this reason, we have adopted the WebAssembly System Interface [97] (or Wasi, henceforth) as the programming model for Veracruz. Intuitively, this can be thought of as "Posix for Wasm", providing a system interface for querying Veracruz's in-memory filesystem, generating random bytes, and executing other similar system tasks. (In this light, the Veracruz runtime can be seen as a simple operating system for Wasm.) By adopting Wasi, one may also use existing libraries and standard programming idioms when targeting Veracruz.

Wasi uses **capabilities**, in a similar vein to Capsicum [92], and a program may only use functionality which it has been explicitly authorized to use. The program, π's, capabilities are specified in the global policy, and typically extend to reading inputs, writing outputs, and generating random bytes, constraining the program to act as a pure, randomized, function.

4.3 Ad hoc acceleration

Many potential Veracruz applications make use of common, computationally intensive, or security-sensitive routines: cryptography, (de)serialization, and similar. While these routines could be compiled into Wasm, this may incur a performance penalty compared to optimized native code, and for operations such as cryptography, compilation to Wasm may not preserve security properties such as timing side-channel safety. Rather, it is beneficial to provide a single, efficient, and correct implementation for common use, rather than routines being compiled into Wasm code haphazardly.

In response, we introduced "native modules" providing acceleration for specific tasks which are linked into the Veracruz runtime and invoked from Wasm programs. In benchmarking one such module—the acceleration of (de)serialization of `Json` documents from the `pinecone` binary format—we observe a 35% speed-up when (de)serializing a vector of 10,000 random elements (238s native vs. 375s Wasm). Additional optimization will likely further boost performance.

Given the *ad hoc* nature of these accelerators, their lack of uniformity, and the fact that more will be added over time, invoking them from Wasm is problematic. Extending the Veracruz system interface to incorporate accelerator-specific functionality would take us beyond Wasi, and require the use of support libraries for programming with Veracruz. Instead, we opt for an interface built around **special files** in the Veracruz filesystem, with modules invoked by Wasm programs writing-to and reading-from these files, reusing existing programming idioms and filesystem support in Wasi.

4.4 Threat model

The Veracruz TCB includes the underlying isolate, the Veracruz runtime, and the implementation of the Veracruz proxy attestation service. The host of the Veracruz attestation service must also be trusted by all parties, as must the native attestation services or keys in use. The correctness of the various protocols in use—TLS, platform-specific native attestation, and PSA attestation—must also be trusted.

The Wasm execution engine must also be trusted to correctly execute a binary, so that a computation is faithfully executed according to the published bytecode semantics [80, 93], and that the program is unable to escape its sandbox, damage or spy on a delegate, or have any other side-effect than allowed by the Veracruz sandboxing model. Recent techniques have been developed that use post-compilation verification to establish this trust [48]—we briefly discuss our ongoing experiments in this area in §6. Compiler verification could be used to engender trust in the Wasm execution engine, though we are not aware of any verified, high-performance Wasm interpreters or JITs suitable for use with Veracruz at the time of writing (see [94] for progress toward this, however). Memory issues have been implicated in attacks against isolates in the past [58]—we write Veracruz in Rust in an attempt to avoid this, with the compiler therefore also trusted.

Veracruz does not defend against denial-of-service attacks: the delegate is in charge of scheduling execution, and liveness guarantees are therefore impossible to uphold. A malicious principal can therefore deny others access to a computation's result, or refuse to provision a data input or program, thereby blocking the computation from even starting.

Different isolation technologies defend against different classes of attacker, and as Veracruz supports multiple technologies we must highlight these differences explicitly.

AWS Nitro Enclaves protect computations from the AWS customer running the EC2 instance associated with the isolate. While AWS assures users that isolates are protected from employees and other insiders, these assurances are difficult to validate (and, as silicon manufacturer, AWS and its employees must always be trusted). Our TCB therefore also contains the Nitro hardware, Linux host used inside the isolate, the attestation infrastructure for Nitro Enclaves, and any AWS insiders with access to that infrastructure.

For Arm CCA Realms only the *Realm Management Monitor* (RMM, henceforth), a separation kernel isolating Realms from each other, has access to the memory of a Realm other than the software executing in the Realm itself. Realms are protected from the non-secure hypervisor, and any other software running on the system other than the RMM, and will be protected against a class of physical attacks using memory encryption. Our TCB therefore contains the RMM, the system hardware, Linux host inside the Realm, along with the attestation infrastructure for Arm CCA.

For IceCap our TCB includes the seL4 kernel which we rely on to securely isolate processes from one another, bolstered by a body of machine-checked proofs of the kernel's security and functional correctness (though at present these

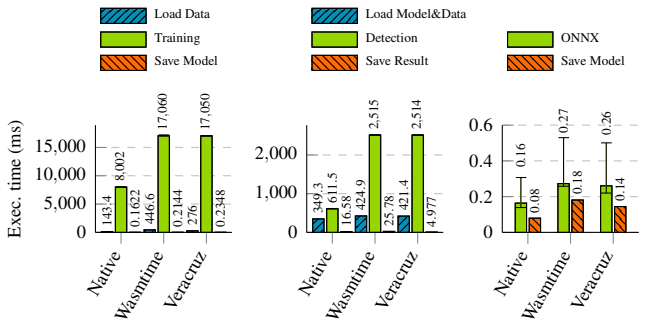

Figure 3: Execution time of the DL examples, classifier training (L), inference (M), and *ONNX* model aggregation (R)

do not extend to the `EL2` configuration for AArch64). For a typical hypervisor deployment of seL4, the SMMU is the only defence against physical attacks.

The TCB of Veracruz includes both local and remote stacks of hardware and software, while purely cryptographic techniques merely rely on a trustworthy implementation of a primitive and the correctness of the primitive itself. As demonstrated in §5, Veracruz provides a degree of efficiency and practicality currently out of reach for purely cryptographic techniques, at the cost of this larger TCB.

Principals face a challenging class of threats stemming from collusion between the other principals, including the delegate. Some algorithms may be particularly vulnerable to an unwanted declassification of secret inputs to any result receiver, and some attacks may be enhanced by collusion between principals—e.g., a side-channel inserted into the program for the benefit of the delegate. As discussed in §2, several powerful side-channel attacks have been demonstrated in the past against software executing within isolates, and other side-channels also exist including wall-clock execution time of the program, π, on the input data sets, and data sizes and arrival times leaked by TLS connections. In cases where programs are secret, principals must trust the program provider not to collude with the result receiver, as a secret program could trivially intentionally leak data into the result or contain convert channels. If the existence of this trust relationship is undesirable, then principals should insist on program declassification before enrolling in a computation.

5 Evaluation

This section uses the following test platforms: Intel Core i7-8700, 16GiB RAM, 1TB SSD (*Core i7*, henceforth); `c5.xlarge` AWS VM, 8GiB RAM, EBS (*EC2*, henceforth); Raspberry Pi 4, 4GiB RAM, 32GB μSD (*RPi4*, henceforth). We use GCC 9.30 for x86-64, GCC 7.5.0 for AArch64, and Wasi SDK-14.0 with LLVM 13.0 for Wasm.

5.1 Case-study: deep learning

Training datasets, algorithms, and learnt models may be sensitive IP and the learning and inference processes are vulnerable to malicious changes in model parameters that can cause a negative influence on a model's behaviors that is hard to detect [10, 62]. We present two Veracruz case-studies in protecting deep learning (DL henceforth) applications: privacy-preserving training and inference, and privacy-preserving model aggregation service, a step toward *federated* DL. We use *Darknet* [63, 78] in both cases, and the *Open Neural Network eXchange* [11, 26] (*ONNX*, henceforth) as the aggregation format. We focus on the *execution time* of training, inference, and model aggregation on the *Core i7* test platform.

In the training and inference case-study, the program receives input datasets from the respective data providers and a pre-learnt model from a model provider. Thereafter, the provisioned program starts training or inference, protected inside Veracruz. The results—that is, the trained model or prediction—are made available to a result receiver. In the model aggregation case-study, clients conduct *local training* with their favorite DL frameworks, convert the models to *ONNX* format, and provision these derived models into Veracruz. The program then aggregates the models, making the result available to all clients. By converting to *ONNX* locally, we support a broad range of local training frameworks—i.e., *PyTorch* [74], *Tensorflow* [1], *Darknet*, or similar.

We trained a LeNet [57] on *MNIST* [57], a dataset of handwritten digits consisting of 60,000 training and 10,000 validation images. Each image is 28×28 pixels and less than 1KiB; we used a batch size of 100 in training, obtaining a trained model of 186KiB. We take the average of 20 trials for training on 100 *batches* (hence, 10,000 images) and then ran inference on one image. For aggregation, we use three copies of this *Darknet* model (186KiB), obtaining three *ONNX* models (26KiB), performing 200 trials for aggregation, as aggregation time is significantly less. Results are presented in Fig. 3.

For all DL tasks we observe the same execution time between Wasmtime and Veracruz, as expected, with both around 2.1–4.1× slower than native CPU-only execution, likely due to more aggressive code optimization available in native compilers. However, the similarity between Wasmtime and Veracruz diverges for file operations such as loading and saving of model data. Loading data from disk is 1.2–3.1× slower when using Wasmtime compared to executing natively. However, I/O in Veracruz is usually *faster* than Wasmtime, and sometimes faster than native execution, e.g., when saving images in inference. This is likely due to Veracruz's in-memory filesystem exhibiting a faster read and write speed transferring data, compared to the SSD of the test machine.

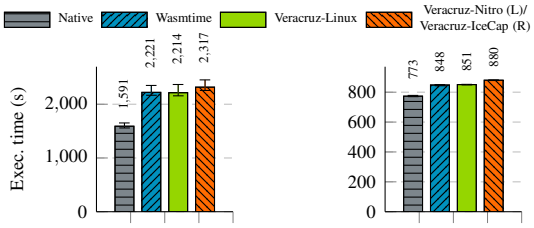

Figure 4: Video object detection execution time on *EC2* (L) and *RPi4* (R)

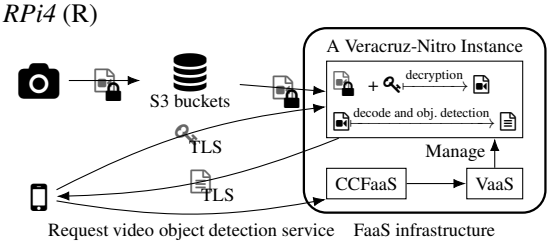

Figure 5: Video object detection case-study

5.2 Case-study: video object detection

We have used Veracruz to prototype a *Confidential FaaS*, running on AWS Nitro Enclaves and using *Kubernetes* [53]. In this model, a cloud infrastructure or other delegate initializes an isolate containing only the Veracruz runtime and provides an appropriate global policy file. Confidential functions are registered in a *Confidential Computing as a Service* (CCFaaS, henceforth) component, which acts as a registry for clients wishing to use the service and which collaborates, on behalf of clients, with a *Veracruz as a Service* (VaaS, henceforth) component which manages the lifetime of any spawned Veracruz instances. Together, the CCFaaS and VaaS components draft policies and initialize Veracruz instances, while attestation is handled by clients, using the proxy attestation service.

Building atop this confidential FaaS infrastructure, we applied Veracruz in a full end-to-end encrypted video object detection flow (see Fig. 5). Our intent is to demonstrate that Veracruz can be applied to industrially-relevant use-cases: here, a video camera manufacturer wishes to offer an object detection service to their customers while providing believable guarantees that they cannot access customer video.

The encrypted video clips originating from an IoTeX Ucam video camera [47] are stored in an *AWS S3* bucket. The encryption key is owned by the camera operator and perhaps generated by client software on their mobile phone or tablet. Independently, a video processing and object detection function, compiled to Wasm, is registered with the CCFaaS component which takes on the role of program provider in the Veracruz computation. This function makes use of the Cisco `openh264` library as well as the Darknet neural network framework and a prebuilt YOLOv3 model, as previously discussed in §5.1, for object detection (our support for Wasi eased this porting).

Upon the request of the camera owner, the CCFaaS and VaaS infrastructure spawn a new AWS Nitro Enclave loaded with the Veracruz runtime, and configured using an appropriate global policy that lists the camera owner as having the role of data provider and result receiver. The confidential FaaS infrastructure forwards the global policy to the camera owner, where it is automatically analyzed by their client software, with the camera owner thereafter attesting the AWS Nitro Enclave instance. If the global policy is acceptable, and attestation succeeds, the camera owner securely connects to the spawned isolate, containing the Veracruz runtime, and securely provisions their decryption key using TLS in their role as data provider. The encrypted video clip is also then provisioned into the isolate, by a dedicated AWS S3 application, which is also listed in the global policy as a data provider, and the computation can then go ahead. Once complete, metadata containing the bounding boxes of any object detected in the frames of the video clips can be securely retrieved by the camera owner via TLS, in their result receiver role, for interpretation by their client software.

Note that in this FaaS infrastructure desirable cloud application characteristics are preserved: the computation is on-demand and scaleable, and our infrastructure allows multiple instances of Veracruz, running different functions, to be executed concurrently. Only the AWS S3 application, the camera owner's client application and the video decoding and object detection function are specific to this use-case. All other modules are generic, allowing other applications to be implemented. Moreover, note that no user credentials or passwords are shared directly with the FaaS infrastructure in realizing this flow, beyond the name of the video clip to retrieve from the AWS S3 bucket and a one-time access credential for the AWS S3 application. Decryption keys are only shared with the Veracruz runtime inside an attested isolate.

We benchmark by passing a 1920×1080 video to the object detection program, which decodes frame by frame, converts, downscales, and passes frames to the ML model. We compare four configurations on two different platforms:

- On *EC2*, a native `x86-64` binary on Amazon Linux; a Wasm binary under Wasmtime-0.27; a Wasm binary inside Veracruz as a Linux process; a Wasm binary inside Veracruz on AWS Nitro Enclaves. The video is 240 frames long and fed to the YOLOv3-608 model [79].
- On *RPi4*: a native AArch64 binary on Ubuntu 18.04 Linux; a Wasm binary under Wasmtime-0.27; a Wasm binary inside Veracruz as a Linux process; a Wasm binary inside Veracruz on IceCap. Due to memory limits the video is 240 frames long and fed to the YOLOv3-tiny model [79].

We take the native `x86-64` configuration as our baseline, and present average runtimes for each configuration, along with observed extremes, in Fig. 4.

Description	Time (ms)
Proxy Attestation Service start	7
Onboard new Veracruz isolate	3122
Request attestation message	54
Initialization of Veracruz isolate	1
Check hashes (including TLS handshake)	184
Provision object detection program	798
Provision data (model, video)	282323

Table 2: Breakdown of Veracruz deployment overheads for the video object detection use-case on AWS Nitro Enclaves

***EC2* results** Wasm (with experimental SIMD support in Wasmtime) has an overhead of ∼39% over native code; most CPU cycles are spent in matrix multiplication, which the native compiler can better autovectorize than the Wasm compiler. The vast majority of execution time is spent in neural network inference, rather than video decode or image downscaling. Since execution time is dominated by the Wasm execution, Veracruz overhead is negligible. A ∼5% performance discrepancy exists between Nitro and Wasmtime, which could originate from our observation that Nitro is slower at loading data into an enclave, but faster at writing, though Nitro runs a different kernel with a different configuration, on a separate CPU, making this hard to pinpoint. Deployment overheads for Nitro are presented in Table 2, showing a breakdown of overheads for provisioning a new Veracruz instance.

***RPi4* results** The smaller ML model significantly improves inference performance at the expense of accuracy. Wasm has an overhead of ∼10% over native code, smaller than the gap on *EC2*, and could be due to reduced vectorization support in GCC's AArch64 backend. Veracruz overhead is again negligible, though IceCap induces an overhead of ∼3% over Veracruz-Linux. This observation approximately matches the overhead of ∼2% for CPU-bound workloads measured in Fig. 1, explained by extra context switching through trusted resource management services during scheduling operations.

Using "native modules", introduced in §4.3, explicit support for neural network inference could be added to the Veracruz runtime, though our results above suggest a max ∼38% performance boost by pursuing this, likely less due to the costs of marshalling data between the native module and Veracruz file system. For larger performance boosts, dedicated ML acceleration could be used, requiring support from the Veracruz runtime, though establishing trust in accelerators outside the isolate is hard, with PCIe attestation still a work-in-progress.

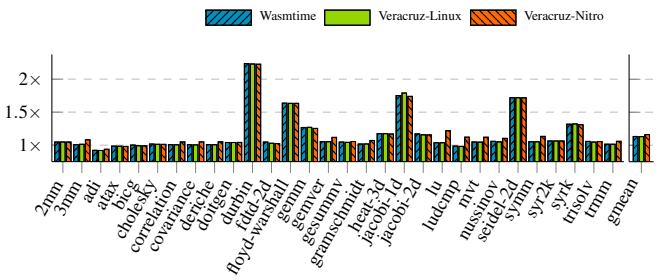

Figure 6: Relative execution time (vs. native) of PolyBench/C (large dataset) on *EC2*. `gmean` shows the geometric mean of all results

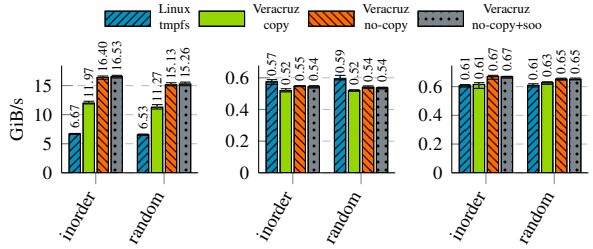

Figure 7: VFS bandwidth: read (L), write (M) and update (R)

5.3 Further comparisons

PolyBench/C microbenchmarks We further evaluate the performance of Veracruz on compute-bound programs using the PolyBench/C suite (version 4.2.1-beta) [75], a suite of small, simple computationally-intensive kernels. We compare execution time of four different configurations on the *EC2* instance running *Amazon Linux 2*: a native `x86-64` binary; a Wasm binary under Wasmtime-0.27; a Wasm binary under Veracruz as a Linux process; and a Wasm binary executing under Veracruz in an AWS Nitro Enclave. We take `x86-64` as our baseline, and present results in Fig. 6. Wasmtime's overhead against native CPU execution is relatively small with a geometric mean of ∼13%, though we observe that some test programs execute even faster under Wasmtime than when natively compiled. Again, we compile our test programs with Wasmtime's experimental support for SIMD proposal, though this boosts performance for only a few programs. Veracruz-Linux doesn't exhibit a visible overhead compared to Wasmtime, which is expected as most execution time is spent in Wasmtime, and the presence of the Veracruz VFS is largely irrelevant for CPU-bound programs. Veracruz-Nitro exhibits a small but noticeable overhead (∼3%) compared to Veracruz-Linux, likely due to the reasons mentioned in §5.2.

VFS performance We evaluate Veracruz VFS I/O performance, previously discussed in §4.2. Performance is measured by timing common granular file-system operations and dividing by input size, to find the expected bandwidth.

Results gathered on *Core i7* test platform with a swap size of zero so that measurements would not be invalidated by physical disk access, are presented in Fig. 7. Here, **read** denotes bandwidth of file read operations, **write** denotes bandwidth of file write operations with no initial file, and **update** denotes bandwidth of file write operations with an existing file. We use two access patterns, in-order and random, to avoid measuring only file-system-friendly access patterns. All random inputs, for both data and access patterns, used reproducible, pseudorandom data generated by `xorshift64` to ensure consistency between runs. All operations manipulate a 64MiB file with 16KiB buffer size—in practice, we expect most files will be within an order of magnitude of this size.

We compare variations of our VFS against Linux's `tmpfs`, the standard in-memory filesystem for Linux. **Veracruz copy** moves data between the Wasm's sandboxed memory and the VFS through two copies, one at the Wasi API layer, and one at the internal VFS API layer. **Veracruz no-copy** improved on this by performing a single copy directly from the Wasm's sandboxed memory into the destination in the VFS. This was made possible thanks to Rust's borrow checker, which is able to express the temporarily shared ownership of the Wasm's sandboxed memory without sacrificing memory or lifetime safety. In theory this overhead can be reduced to zero copies through `memmap`, however this API is not available in standard Wasi. **Veracruz no-copy+soo** is our latest design, extending the no-copy implementation with a small-object optimization (SOO) `iovec` implementation—a Wasi structure describing a set of buffers containing data to be operated on, which for the majority of operations contain a reference to a single buffer. Through this, we inline two or fewer buffers into the `iovec` structure itself, completely removing memory allocations from the read and write path for all programs we tested with. Performance impact is negligible, however.

Being in an-memory filesystem, the internal representation is relatively simple: directories and a global `inode` table are implemented using hash tables, with each file represented as a vector of bytes. While apparently naïve, these data-structures have seen decades of optimization for in-memory performance, and even sparse files perform efficiently due to RAM over-commitment by the runtimes. However, we were still surprised to see very close performance between Veracruz and `tmpfs`, with Veracruz nearly doubling the `tmpfs` performance for reads, likely due to the overhead of kernel syscalls necessary to communicate with `tmpfs` in Linux. (Unfortunately `tmpfs` is deeply integrated into the Linux VFS layer, so it is not possible to compare with `tmpfs` in isolation.)

Both Veracruz and `tmpfs` use hash tables to store directory information, with the file data-structure and memory allocator representing significant differences. In Veracruz we use byte vectors backed by the runtime's general purpose allocator, whereas `tmpfs` uses a tree of pages backed by the Linux

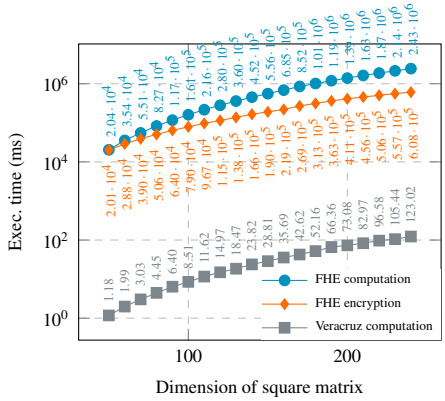

Figure 8: SEAL and Veracruz computation performance

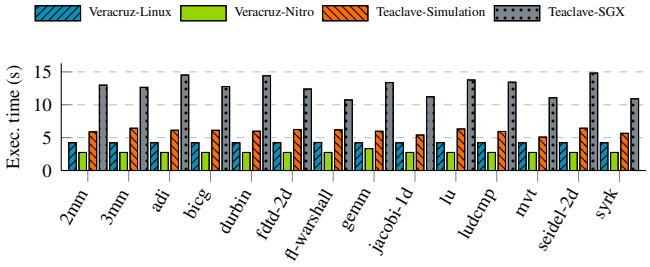

Figure 9: Execution times of Veracruz and Apache Teaclave

VFS's page cache, which acts as a cache-aware fixed-size allocator. We expect this page cache to have a much cheaper allocation cost, at the disadvantage of storing file data in non-linear blocks of memory—observable in the difference between the **write** and **update** measurements. For **write**, `tmpfs` outperforms Veracruz due to faster memory allocations and no unnecessary copies, while **update** requires no memory allocation, and has more comparable performance.

Fully-homomorphic encryption An oft-suggested use-case for fully-homomorphic encryption (FHE, hencefoth) is protecting delegated computations. We briefly compare Veracruz against SEAL [61], a leading FHE library, in computing a range of matrix multiplications over square matrices of various dimensions. Algorithms in both cases are written in `C`, though floating point arithmetic is replaced by the SEAL multiplication function for use with FHE. Results are presented in Fig. 8. Our results demonstrate that overheads for FHE are impractical, even for simple computations.

Teaclave Apache Teaclave [33] is a privacy-preserving FaaS infrastructure built on Intel SGX, supporting Python and Wasm with a custom programming model using the Wamr [3] interpreter. We compare the performance of Teaclave running under Intel SGX with Veracruz as a Linux process, both on *Core i7*, and Veracruz on AWS Nitro enclaves on *EC2*—admittedly an imperfect comparison, due to significant differences in design, isolation technology, Wasm

runtime, and hardware between the two. We run the PolyBench/C suite with its mini dataset—Teaclave's default configuration errors for larger datasets—and measure end-to-end execution time, which includes initialization, provisioning, execution and fetching the results, which we present in Fig. 9. While Veracruz has better performance than Teaclave when executing Wasm—with Veracruz under AWS Nitro exhibiting a mean 2.11× speed-up compared to Teaclave in simulation mode, and faster still than Teaclave in SGX—the fixed initial overhead of Veracruz, ∼4s in Linux and ∼2.7s in AWS Nitro, dominates the overall overhead in either case.

6 Closing remarks

We have introduced *Veracruz*, a framework for designing and deploying privacy-preserving delegated computations among a group of mutually mistrusting principals, using isolates as a "neutral ground" to protect computations from prying or interference. In addition to supporting a number of hardware-backed Confidential Computing technologies—such as AWS Nitro Enclaves and Arm Confidential Computing Architecture Realms—Veracruz also supports pragmatic "software isolates" through *IceCap*. IceCap makes use of the high-assurance seL4 microkernel, on Armv8-A platforms without any other explicit support for Confidential Computing, to provide strong isolation guarantees for virtual machines.

Veracruz, with IceCap, provides a uniform programming and attestation model across emerging and "legacy" hardware platforms, easing the deployment of delegated computations. Both projects are open-source [45, 89], and Veracruz is adopted by the LF's *Confidential Computing Consortium*.

Related work Isolates have been used to protect a zoo of computations of interest, e.g., ML [20, 54, 72, 83, 85] and genomic computations [21, 56, 59], and have been used to emulate or speed up cryptographic techniques such as functional encryption [36] and secure multi-party computations [35, 40, 76]. These can be seen as use-cases, specialized with a particular policy and program, of Veracruz.

OpenEnclave [73] provides a common development platform for SGX Enclaves and TrustZone trusted applications. Veracruz provides a higher-level of abstraction than OpenEnclave, and includes various support libraries, client code, and attestation protocols to ease the provisioning of programs into an isolate. Veracruz also supports a wider range of isolates, including both hardware- and software-isolates.

Previous work [52] suggested a framework similar to Veracruz, but never implemented it. *Google Oak* [39], *Profian Enarx* [77], *Apache Teaclave* [5], *Fortanix Confidential Computing Manager* [37] and *SCONE* [8] are similar to Veracruz, though significant differences exist. Oak's emphasis is information flow control, while Enarx, Fortanix, and SCONE protect the integrity of legacy computations, either requiring recompilation to Wasm, or supporting containerized workloads under SGX, respectively. Apache Teaclave is the most similar project, discussed in §5, and we perform significantly better. The proxy attestation service, and our certificate-based attestation protocol, especially suitable for clients on resource-constrained devices, is also unique.

Protected KVM (pKVM) [31, 34] is an attempt to minimize the TCB of KVM, enabling virtualization-based confidential computing on mobile platform, and similar in spirit to IceCap. pKVM, with an EL2 kernel specifically designed for the task, may have higher performance than IceCap, but will not benefit from the formal verification effort invested in seL4.

OPERA [23] places a proxy between client code and the Intel Attestation Service, exposing the same EPID protocol to clients as the web-service exposes. The Veracruz proxy exposes a potentially different protocol to client code, compared to the native protocol, due to the variety of isolates Veracruz supports. Intel's *Data Center Attestation Primitives (DCAP)*, also serves similar use-cases, reducing the number of calls to an external attestation service when authenticating attestation tokens, though is limited to use with Intel SGX.

Ongoing and future work The proxy attestation service, which currently signs each generated certificate with the same key, could sign certificates for different isolation technologies with different keys, each associated with a different root CA certificate. With this, a global policy could choose which technology to support based on the selection of root CA certificate embedded in the policy, and if multiple isolation technologies were to be supported, more than one root CA certificate could be embedded. The proxy attestation server could also maintain multiple Root CA certificates, arranged into a "decision tree of certificates", with the server choosing a CA certificate to use when signing the isolate's certificate from the tree, following a path from the root described by characteristics of the isolate technology itself (e.g., name of the manufacturer, whether memory encryption is supported, and so on). Again, the certificate associated with the security profile of the desired isolation technology can be embedded in the policy.

We also aim to bound the intensional and extensional properties of programs provisioned into Veracruz. Pragmatically, cryptographic operations are perhaps most sensitive to timing attacks, and we aim to provide a limited defense by supplying a constant-time cryptography implementation—using `mbedtls` [60]—via the native module facility discussed in §4.3. Moreover, we aim to explore the use of a statically verified, constant-time virtual machine to gives users the option to statically verify timing properties of their programs—an area of significant recent academic interest—though likely at the cost of limiting their program to constant-time constructs, which is intractable for general-purpose programming. Us-

ing FaCT [19] Veracruz could provide flexible, verifiably constant-time components such as virtual machines or domain specific functions, while the CT-Wasm [95] extension for Wasm also provides verifiable, constant-time guarantees as a set of secrecy-aware types and bytecode instructions. CT-Wasm has not yet adopted by the Wasm committee.

We are also continuing work on statically verifying the *Software Fault Isolation* (SFI, henceforth) safety of sandboxed applications. SFI systems, such as Wasm, add runtime checks to loads, stores, and control flow transfers to ensure sandboxed code cannot escape from its address space region, though bugs in SFI compilers can (and do) incorrectly remove these checks and introduce bugs that let untrusted code escape its sandbox [12, 43]. To address this—following other SFI systems [65, 99, 102]—we have built a static verifier for binary code executed by Veracruz, implemented as an extension of *VeriWasm* [49], an open-source SFI verifier for compiled Wasm code. To adapt *VeriWasm* to Veracruz, we added support for AArch64, and ported VeriWasm from the *Lucet* [17] toolchain to Wasmtime, as used by Veracruz. We plan to further extend VeriWasm to check other properties besides software fault isolation, e.g., Spectre [70] resistance.

Finally, observe that the provisioned program, π, is either kept classified by its owner, or is declassified to a subset of the other principals in the computation (maybe all). In the former case, other principals either must either implicitly trust that π behaves in a particular way, or establish some other mechanism bounding the behavior of the program, out-of-band of Veracruz. We aim for a middle ground, allowing a program owner to declassify runtime *properties* of the program, enforced by Veracruz, while retaining secrecy of the program binary (using e.g., [66]).

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In *NSDI*, 2020.

[3] Bytecode Alliance. WebAssembly Micro Runtime main development repository. https://github.com/bytecodealliance/wasm-micro-runtime. Accessed 2022-02-01.

[4] D.P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, volume 1, pages 73–80, 2006.

[5] Apache Teaclave main development repository. https://github.com/apache/incubator-teaclave. Accessed 2022-01-26.

[6] Arm Confidential Compute Architecture (Arm CCA). https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture. Accessed 2022-01-20.

[7] Arm TrustZone technology for Cortex-A and Cortex-M. https://developer.arm.com/ip-products/security-ip/trustzone. Accessed 2022-01-26.

[8] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, André Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark Stillwell, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Peter R. Pietzuch, and Christof Fetzer. SCONE: Secure Linux containers with Intel

SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 689–703, 2016.

[9] AWS Nitro Enclaves user guide: Cryptographic attestation. https://docs.aws.amazon.com/enclaves/latest/user/set-up-attestation.html. Accessed 2022-01-12.

[10] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[11] Junjie Bai, Fang Lu, Ke Zhang, et al. ONNX: the Open Neural Network Exchange format. https://github.com/onnx/onnx. Accessed 2022-01-24.

[12] Alexandre Bartel and John Doe. Twenty years of escaping the Java sandbox. In *Phrack*, 2018.

[13] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software Grand Exposure: SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, August 2017. USENIX Association.

[14] Ernie Brickell and Jiangtao Li. Enhanced Privacy ID from bilinear pairing for hardware authentication and attestation. In *2010 IEEE Second International Conference on Social Computing*, pages 768–775, 2010.

[15] Ernie Brickell and Jiangtao Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. *IEEE Trans. Dependable Secur. Comput.*, 9(3):345–360, 2012.

[16] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1041–1056, Vancouver, BC, August 2017. USENIX Association.

[17] Bytecode Alliance. Lucet. https://github.com/bytecodealliance/lucet. Accessed 2022-01-25.

[18] The CapDL domain specific language documentation. https://docs.sel4.systems/projects/capdl/. Accessed 2022-01-25.

[19] Sunjay Cauligi, Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad S. Wahby, John Renner, Benjamin Grégoire, Gilles Barthe, Ranjit Jhala, and Deian Stefan. FaCT: A DSL for timing-sensitive computation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 174–189, New York, NY, USA, 2019. Association for Computing Machinery.

[20] Swarup Chandra, Vishal Karande, Zhiqiang Lin, Latifur Khan, Murat Kantarcioglu, and Bhavani Thuraisingham. Securing data analytics on SGX with randomization. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security – ESORICS 2017*, pages 352–369, Cham, 2017. Springer International Publishing.

[21] Feng Chen, Chenghong Wang, Wenrui Dai, Xiaoqian Jiang, Noman Mohammed, Md Momin Al Aziz, Md Nazmus Sadat, Cenk Sahinalp, Kristin Lauter, and Shuang Wang. PRESAGE: PRivacy-preserving gEnetic testing via SoftwAre Guard Extension. *BMC Med Genomics*, 10(Suppl 2):48, Jul 2017.

[22] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. SgxPectre: Stealing Intel secrets from SGX Enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157, 2019.

[23] Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. OPERA: Open remote attestation for Intel's secure enclaves. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2317–2331, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D. Garcia. VoltPillager: Hardware-based fault injection attacks against Intel SGX enclaves using the SVID voltage scaling interface. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 699–716. USENIX Association, August 2021.

[25] The Linux Foundation's Confidential Computing Consortium (CCC) homepage. https://confidentialcomputing.io. Accessed 2022-01-27.

[26] cONNXr: a pure C runtime for ONNX. https://github.com/alrevuelta/cONNXr. Accessed 2022-01-24.

[27] Lucian Constantin. Intel SGX users need CPU microcode patch to block PLATYPUS secrets-leaking attack. https://www.csoonline.com/article/3596564/intel-sgx-users-need-cpu-microcode-patch-to-block-pl html. Accessed 2022-01-24.

[28] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 5280, RFC Editor, May 2008.

[29] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016, 2016.

[30] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. Cachequote: Efficiently recovering long-term secrets of SGX EPID via cache attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):171–191, 2018.

[31] Will Deacon. Virtualization for the masses: Exposing KVM on Android. In *The KVM Forum*, 2022. Accessed 2022-01-28.

[32] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Sixth Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150, San Francisco, CA, 2004.

[33] Ran Duan, Long Li, Chan Zhao, Shi Jia, Yu Ding, Yulong Zhang, Huibo Wang, Yueqiang Cheng, Lenx Wei, and Tanghui Chen. Rust SGX SDK. https://github.com/apache/incubator-teaclave-sgx-sdk, Jun 2020. Accessed 2020-04-15.

[34] Jake Edge. KVM for Android. https://lwn.net/Articles/836693/, 2020. Accessed 2022-01-27.

[35] Susanne Felsen, Ágnes Kiss, Thomas Schneider, and Christian Weinert. Secure and private function evaluation with Intel SGX. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, CCSW'19, page 165–181, New York, NY, USA, 2019. Association for Computing Machinery.

[36] Ben Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, and Sergey Gorbunov. IRON: Functional encryption using Intel SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 765–782, New York, NY, USA, 2017. Association for Computing Machinery.

[37] Fortanix Confidential Computing Manager homepage. https://support.fortanix.com/hc/en-us. Accessed 2022-01-27.

[38] Qian Ge, Yuval Yarom, Tom Chothia, and Gernot Heiser. Time protection: The missing os abstraction. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.

[39] Google project Oak. https://github.com/project-oak/oak. Accessed 2020-04-15.

[40] Debayan Gupta, Benjamin Mood, Joan Feigenbaum, Kevin Butler, and Patrick Traynor. Using Intel Software Guard Extensions for efficient two-party secure function evaluation. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 302–318, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[41] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, page 185–200, New York, NY, USA, 2017. Association for Computing Machinery.

[42] Shai Halevi. Advanced cryptography: Promise and challenges. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, CCS '18, page 647, New York, NY, USA, 2018. Association for Computing Machinery.

[43] L. Hansen. Mark the `jump_table_entry` instruction as loading. https://github.com/bytecodealliance/cranelift/pull/805. Accessed 2022-01-25.

[44] Tianlin Huo, Xiaoni Meng, Wenhao Wang, Chunliang Hao, Pei Zhao, Jian Zhai, and Mingshu Li. Bluethunder: A 2-level directional predictor based side-channel attack against SGX. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):321–347, 2020.

[45] The IceCap development repository. URL redacted for double-blind review. Accessed 2022-01-27.

[46] Intel Trust Domain Extensions (Intel TDX): White paper (v4). https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf. Accessed 2022-01-25.

[47] Ucam. https://ucam.iotex.io/. Accessed 2022-01-27.

[48] Evan Johnson, David Thien, Yousef Alhessi, Shravan Narayan, Fraser Brown, Sorin Lerner, Tyler McMullen, Stefan Savage, and Deian Stefan. Доверя́й,

но проверя́й: SFI safety for native-compiled Wasm. In *NDSS*. Internet Society, 2021.

[49] Evan Johnson, David Thien, Yousef Alhessi, Shravan Narayan, Fraser Brown, Sorin Lerner, Tyler McMullen, Stefan Savage, and Deian Stefan. Trust, but verify: SFI safety for native-compiled Wasm. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2021.

[50] David Kaplan, Jeremy Powell, and Tom Woller. AMD memory encryption: white paper (v7). https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf, 2016. Accessed 2022-01-25.

[51] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. KVM: the Linux virtual machine monitor. In *In Proceedings of the 2007 Ottawa Linux Symposium (OLS'-07*, 2007.

[52] Patrick Koeberl, Vinay Phegade, Anand Rajan, Thomas Schneider, Steffen Schulz, and Maria Zhdanova. Time to rethink: Trust brokerage using Trusted Execution Environments. In Mauro Conti, Matthias Schunter, and Ioannis G. Askoxylakis, editors, *Trust and Trustworthy Computing - 8th International Conference, TRUST 2015, Heraklion, Greece, August 24-26, 2015, Proceedings*, volume 9229 of *Lecture Notes in Computer Science*, pages 181–190. Springer, 2015.

[53] The Kubernetes project homepage. https://kubernetes.io/docs/home/. Accessed 2022-01-25.

[54] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. TensorSCONE: a secure TensorFlow framework using Intel SGX. *CoRR*, abs/1902.04413, 2019.

[55] Ihor Kuz, Gerwin Klein, Corey Lewis, and Adam Walker. capDL: A language for describing capability-based systems. In *Proceedings of the* 1st *ACM Asia-Pacific Workshop on Systems (APSys)*, pages 31–36, 06 2010.

[56] C. Lambert, M. Fernandes, J. Decouchant, and P. Esteves-Verissimo. MaskAl: Privacy preserving masked reads alignment using Intel SGX. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 113–122, 2018.

[57] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[58] Jaehyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado, and Brent Byunghoon Kang. Hacking in darkness: Return-oriented programming against secure enclaves. In *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, page 523–539, USA, 2017. USENIX Association.

[59] Avradip Mandal, John C. Mitchell, Hart Montgomery, and Arnab Roy. Data oblivious genome variants search on Intel SGX. In Joaquin Garcia-Alfaro, Jordi Herrera-Joancomartí, Giovanni Livraga, and Ruben Rios, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 296–310, Cham, 2018. Springer International Publishing.

[60] MbedTLS cryptography library. https://www.trustedfirmware.org/projects/mbed-tls/. Accessed 2022-01-25.

[61] The Microsoft SEAL fully-homomorphic encryption library development repository (v3.7). https://github.com/Microsoft/SEAL. Accessed 2022-01-26.

[62] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 94–108, 2021.

[63] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 161–174, 2020.

[64] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. Cachezoom: How SGX amplifies the power of cache attacks. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 10529 of *Lecture Notes in Computer Science*, pages 69–90. Springer, 2017.

[65] Greg Morrisett, Gang Tan, Joseph Tassarotti, Jean-Baptiste Tristan, and Edward Gan. RockSalt: better, faster, stronger SFI for the x86. In *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*, pages 395–404, 2012.

[66] Dominic P. Mulligan and Nick Spinale. The Supervisionary proof-checking kernel, or: a work-in-progress

towards proof-generating code (extended abstract). https://dominicpm.github.io/publications/mulligan-supervisionary-2022.pdf, 2022.

[67] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against Intel SGX. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[68] Toby C. Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, Timothy Bourke, Sean Seefried, Corey Lewis, Xin Gao, and Gerwin Klein. sel4: From general purpose to a proof of information flow enforcement. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 415–429, 2013.

[69] Toby C. Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, and Gerwin Klein. Noninterference for operating system kernels. In *Certified Programs and Proofs - Second International Conference, CPP 2012, Kyoto, Japan, December 13-15, 2012. Proceedings*, pages 126–142, 2012.

[70] Shravan Narayan, Craig Disselkoen, Daniel Moghimi, Sunjay Cauligi, Evan Johnson, Zhao Gang, Anjo Vahldiek-Oberwagner, Ravi Sahita, Hovav Shacham, Dean Tullsen, and Deian Stefan. Swivel: Hardening WebAssembly against Spectre. In *USENIX Security Symposium*. USENIX, August 2021.

[71] M. Nystrom and B. Kaliski. PKCS #10: Certification request syntax specification version 1.7. RFC 2986, RFC Editor, November 2000.

[72] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 619–636, USA, 2016. USENIX Association.

[73] The OpenEnclave development repository. https://github.com/openenclave/openenclave. Accessed 2022-01-27.

[74] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[75] The PolyBench/C benchmarking suite homepage. http://web.cs.ucla.edu/~pouchet/software/polybench/. Accessed 2022-01-28.

[76] Bernardo Portela, Manuel B M Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. Secure multiparty computation from SGX. In *Financial Cryptography and Data Security 2017*. International Financial Cryptography Association, April 2017.

[77] Profian Enarx development repository. https://github.com/enarx/enarx. Accessed 2022-01-26.

[78] Joseph Redmon. Darknet: open source neural network framework written in C and CUDA. https://github.com/pjreddie/darknet, 2013–2016. Accessed 2022-01-24.

[79] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv*, 2018.

[80] Andreas Rossberg, Ben L. Titzer, Andreas Haas, Derek L. Schuff, Dan Gohman, Luke Wagner, Alon Zakai, J. F. Bastien, and Michael Holman. Bringing the web up to speed with WebAssembly. *Commun. ACM*, 61(12):107–115, 2018.

[81] seL4 inter-process communication (IPC) documentation. https://docs.sel4.systems/Tutorials/ipc.html. Accessed 2022-01-25.

[82] Thomas Sewell, Simon Winwood, Peter Gammie, Toby C. Murray, June Andronick, and Gerwin Klein. sel4 enforces integrity. In *Interactive Theorem Proving - Second International Conference, ITP 2011, Berg en Dal, The Netherlands, August 22-25, 2011. Proceedings*, pages 325–340, 2011.

[83] Fahad Shaon, Murat Kantarcioglu, Zhiqiang Lin, and Latifur Khan. SGX-BigMatrix: A practical encrypted data analytic framework with trusted processors. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1211–1228, New York, NY, USA, 2017. Association for Computing Machinery.

[84] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the perils of Security-Oblivious energy management. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1057–1074, Vancouver, BC, August 2017. USENIX Association.

[85] Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[86] H. Tschofenig, S. Frost, M. Brossard, A. Shaw, and T. Fossati. Arm's Platform Security Architecture (PSA) attestation token, Nov 2019. Accessed 2020-04-15.

[87] Jo Van Bulck, Frank Piessens, and Raoul Strackx. SGX-Step: A practical attack framework for precise enclave execution control. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, SysTEX'17, New York, NY, USA, 2017. Association for Computing Machinery.

[88] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 178–195, New York, NY, USA, 2018. Association for Computing Machinery.

[89] The Veracruz development repository. URL redacted for double-blind review. Accessed 2022-01-27.

[90] The WASMI WebAssembly interpreter. https://docs.rs/wasmi. Accessed 2022-01-27.

[91] Wasmtime: a just-in-time compiler for WebAssembly. https://wasmtime.dev. Accessed 2020-04-15.

[92] Robert N. M. Watson, Jonathan Anderson, Ben Laurie, and Kris Kennaway. A taste of Capsicum: practical capabilities for UNIX. *Commun. ACM*, 55(3):97–104, 2012.

[93] Conrad Watt. Mechanising and verifying the WebAssembly specification. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 53–65, 2018.

[94] Conrad Watt, Xiaojia Rao, Jean Pichon-Pharabod, Martin Bodin, and Philippa Gardner. Two mechanisations of WebAssembly 1.0. In *Proceedings of the 24th international symposium of Formal Methods (FM21), Beijing, China; November 20-25, 2021*, 2021.

[95] Conrad Watt, John Renner, Natalie Popescu, Sunjay Cauligi, and Deian Stefan. CT-Wasm: Type-driven secure cryptography for the Web ecosystem. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[96] The WebAssembly project homepage. https://webassembly.org/. Accessed 2022-01-20.

[97] The WebAssembly System Interface (Wasi) homepage. https://wasi.dev. Accessed 2022-01-27.

[98] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656, 2015.

[99] Bennet Yee, David Sehr, Gregory Dardyk, J Bradley Chen, Robert Muth, Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar. Native client: A sandbox for portable, untrusted x86 native code. In *2009 30th IEEE Symposium on Security and Privacy*, pages 79–93. IEEE, 2009.

[100] The Zephyr project homepage. https://www.zephyrproject.org/. Accessed 2022-01-31.

[101] Ning Zhang, Kun Sun, Deborah Shands, Wenjing Lou, and Y. Thomas Hou. Trusense: Information leakage from TrustZone. In *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*, pages 1097–1105. IEEE, 2018.

[102] Lu Zhao, Guodong Li, Bjorn De Sutter, and John Regehr. Armor: fully verified software fault isolation. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 289–298, 2011.

This figure "icecap-diagram.png" is available in "png" format from:

http://arxiv.org/ps/2205.03322v1