Research Statement: Ensuring Compliance with Data Privacy and Usage Policies

Aastha Mehta
www.mpi-sws.org/~aasthakm

The massive collection and aggregation of personal information via social media, e-commerce websites, mobile applications, etc., brings with it the risk of unintended disclosure and misuse of users’ information. Such incidents abound and can lead to embarrassment\(^1\), discrimination\(^2\), financial loss\(^3\), risks to personal safety\(^4\), and even undermine society’s political discourse\(^5\). Legislators have reacted to this threat with stricter laws and regulations, such as the EU GDPR\(^6\) and CCPA\(^7\).

What remains is the technical challenge for service providers large and small to ensure that their complex and rapidly evolving systems remain compliant with laws, regulations, policies, and users’ choices. To address this challenge:

> **My goal is to build practical systems that ensure compliance by default with a broad class of data privacy and usage policies in online services.**

However, ensuring compliance in modern services is difficult. Retrofitting compliance checks within existing large and rapidly evolving business logic is neither safe nor a scalable approach, as any bug could undermine compliance. Such bugs have caused accidental, large-scale data disclosures and misuse\(^8\). For example, a bug in Facebook’s Photo API exposed millions of private photos to third-party apps\(^9\). To prevent such accidental disclosures and misuse, I have worked on developing policy compliance systems that protect data despite application bugs and misconfigurations.

Specifically, I have developed systems that protect data in persistent storage systems (EuroSys’15) \(^1\), distributed applications (USENIX Security’16) \(^2\), and database-backed applications (USENIX Security’17) \(^3\).

In addition to getting direct access to sensitive data due to accidental compliance failures, an adversary may also indirectly infer the sensitive data via side channels, even if a provider follows good security practices. Side-channel disclosures arise when an adversary shares a physical resource with the victim application, observes the application’s use of the shared resource, and infers the application’s sensitive data from its usage pattern. For instance, an adversary may learn the content of a VoIP conversation by observing just the sizes and timing of packets transmitted via shared network elements. In my research, I have worked on building systems that protect data in Cloud applications against side channels, specifically memory (USENIX Security’16) \(^4\) and network side channels (under submission) \(^5\).

The central theme of my research is building practical systems that allow specifying and enforcing policies separately from application logic, therefore decoupling compliance from application correctness and ensuring all application accesses to data are compliant by default. I empirically evaluate the security guarantees and performance overheads of each system, and optimize the system’s performance to make it practical and usable. To this end, I use principles from security, operating systems, distributed systems, and networking.

1. **Mitigating accidental data disclosures and misuse**

Data processing systems handle data with diverse and complex privacy and usage policies. For instance, user data in online social networks may be limited to friends, user’s click history may be used only for personalization and may require expiration, email is private to its sender/recipient, and a company’s personnel records may be accessible in full only to HR (human resources) employees while a subset may be accessible to the remaining employees.

Contemporary systems attempt to enforce policies by scattering data access checks throughout the application codebase in an ad-hoc manner. Ensuring continued compliance in this manner is cumbersome and error-prone. As applications evolve or data policies change, developers potentially need to revisit all code paths to ensure that all data accesses continue to remain compliant. It is easy to miss some checks or implement incorrect checks, which can subsequently cause unintended disclosures or misuse, indeed, such incidents continue to affect social networking, healthcare, and many other services.

To systematically prevent such compliance failures, I believe that policy compliance must be separated from the core business logic of the applications. Specifically, policies must be specified directly on the data to be protected, and they must be enforced independently of the application code by intercepting the application’s data accesses at appropriate points in the software stack. To realize this goal, several questions need to be addressed: What constitutes a good data abstraction to associate policies with? What constitutes a good policy specification language? At what layer in the software stack should data accesses be intercepted? How can the overheads due to policy enforcement be minimized? I have addressed these questions by building three systems—Qapla \(^3\), Guardat \(^1\), and Thoth \(^2\)—that enable compliance in different classes of applications. These complementary systems cover different tradeoffs between security, performance, and usability.

\(^1\)Embarrassing photos, \(^2\)Price discrimination, \(^3\)Exploiting stolen credit cards, \(^4\)Robbers check social media, \(^5\)Cambridge Analytica, \(^6\)General Data Protection Regulation, \(^7\)California Consumer Privacy Act, \(^8\)Number of Breach Incidents by Source, \(^9\)Facebook Photo API bug
Database-backed applications. Qapla [3] facilitates policy compliance in applications, such as personnel management systems, that are backed by relational databases (e.g., MySQL). It supports a rich set of expressive, fine-grained, and complex policies on database data—including policies mandated by GDPR or user preferences. Policies may constrain who can access which data (rows, columns, and cells) under what conditions (e.g., time, user role); they may also limit which data can be linked together or aggregated.

Qapla’s design has four key aspects: (i) Policies are specified in a declarative language and associated directly with the database schema. Such a centralized and declarative specification allows policies to be designed independent of the application logic, and also makes it easier to reason about and audit the policies. (ii) Policy checks follow a whitelist principle: an application is allowed to access data only if it is subject to an explicitly written policy, and if the policy check succeeds at runtime. The whitelist principle ensures that data can never leak due to accidental omission of policies. (iii) Policies are enforced at runtime by the database adapter, which is outside the application and intercepts all application accesses to the data. Application code is, therefore, not trusted for policy enforcement and policies are enforced even if the application misses policy checks or makes incorrect policy checks. (iv) Qapla is transparent to policy-compliant application queries, and returns the policy-compliant subset of results for non-compliant queries.

In summary, Qapla is a practical system that preserves maximal application functionality while ensuring compliance by default. I demonstrated Qapla’s effectiveness by using it to ensure compliance in two web applications: the HotCRP conference management system, and our institute’s job application portal.

Storage devices and distributed applications. Similar to Qapla, Guardat [1] and Thoth [2] also enable declarative specification of policies directly on the data. However, these systems specify and enforce policies on a different data abstraction, and trade off granularity of the policies they support for stronger security properties and higher efficiency. Guardat specifies confidentiality, integrity, and access accounting policies on coarse-grained files on a traditional block storage device. Guardat enforces these policies directly in the storage controller, thus protecting the persistent state even from a compromised OS and hypervisor. Thoth extends policy compliance to applications designed as distributed data-processing pipelines (e.g., a search engine). It extends the data policies with constraints on how data can flow between different application components. To ensure compliance with these extended policies, Thoth tracks the flow of data through conduits (e.g., files, sockets, pipes, key-value tuples), and enforces policies at the boundary of each application process.

In summary, Qapla, Guardat, and Thoth demonstrate that separating policy specification and enforcement from applications helps to ensure compliance without requiring correctness of the entire application.

2 Mitigating side-channel disclosures

Side channels arise when physical resources are shared among mutually untrusting parties. By observing an application’s usage pattern of a shared resource, which may be correlated with its secrets, an adversary can infer the application’s secrets. Numerous side-channel disclosures have been demonstrated in public Clouds in recent years, which exploit a variety of resources (e.g., CPUs, caches, and memory) shared between tenants co-located on the same server or between the tenants and the host (hypervisor and OS). Additionally, the network, which is shared both within the Cloud and in the public Internet, has also been exploited for side-channel disclosures. These side channels have been demonstrated to recover cryptographic keys and sensitive data from various applications.

Mitigating side-channel disclosures requires: (i) identifying the channel over which the leak occurs and the vector of leak, i.e., the metadata that is correlated with sensitive information, and then (ii) systematically dissociating sensitive information that is observable on the channel from the vector of leak.


Network side-channel mitigation for Cloud services. A network side-channel disclosure involves two steps. First, an adversary observes the shape of an application’s traffic, which may include attributes such as the number, sizes, and timing of network packets. This can be done by an adversary either (i) directly, if it has access to network elements like switches or routers (e.g., an ISP); or (ii) indirectly, if it does not have access to network elements (e.g., a Cloud tenant), by generating cross-traffic that shares bandwidth with the application’s traffic at the server’s network card or a switch. Secondly, from the traffic shape, the adversary can then infer the application’s secrets. Traffic shape has been shown to reveal videos streamed, VoIP call content, cryptographic keys, and even more sensitive information like users’ medical conditions or income.

My work proposes shaping the application’s traffic so that its observable attributes are decorrelated from its secrets. Shaping involves padding all outgoing messages to a secret-independent size, and transmitting all packets at secret-independent times. If done naively, however, shaping can be very costly in terms of bandwidth or latency when the payload traffic is bursty. To make shaping more efficient, my work allows the shape of the application’s traffic to vary based on public information, while still hiding private information. For instance, if the type of content being requested from a server...
(e.g., document vs. video) is not a secret, then different traffic shapes can be used for the two content types. The choice of public parameters thus determines the tradeoff between the privacy afforded to the application and the network overhead incurred due to traffic shaping.

To enforce a traffic shape, I propose the abstraction of a cloaked tunnel, which encapsulates and shapes traffic between each pair of an application’s network endpoints. Pacer [5] realizes the cloaked tunnel abstraction for the Cloud environment, where an unprivileged adversary may be co-located with a tenant’s application on the same server and therefore may observe the tenant’s traffic at the shared network interface of the server. To handle this case, the tunnel is integrated with the server hosting the tenant’s VMs.

Implementing the tunnel involves several challenges. A key challenge is that the application’s secrets can introduce timing leaks into the network traffic by interfering with Pacer’s execution. Such timing leaks may manifest through: (i) the sizes of packets generated by the TCP stack, or (ii) through CPU microarchitectural state and contention on memory and PCI buses, which can affect the timing of Pacer’s transmissions in the tunnel. I implemented Pacer to mask such timing leaks, thus performance-isolating it from the application’s secrets. To overcome the overheads of achieving this isolation, I also designed a secure packet batching technique to allow Pacer to retain a line rate close to that of the NIC.

In summary, Pacer’s tunnel provides a network side-channel mitigation that is secure by design and construction; implementing the tunnel requires modest changes to the Cloud hypervisor and the guest OS, and minimal support from guest applications. Empirically, Pacer can shape traffic of web and video services with only modest overheads on bandwidth, client latencies, and server throughput.

**Network side-channel mitigation in the public Internet.** Network side-channel disclosures are relevant even for services hosted within corporate premises and communicating via a secure VPN over the public Internet (e.g., VoIP service). Unlike in the Cloud setting, an adversary in this setting observes the VPN traffic only at network elements in the public Internet. In my ongoing work [6], I am building the cloaked tunnel’s endpoints as middleboxes that can be integrated with gateway nodes of the corporate network. This approach enables me to avoid modifications to the OS on application servers and provide mitigation for applications with minimal effort from the application developers and system administrators. However, while Cloud services typically follow a request-response pattern, VoIP services involve traffic that is highly dynamic in time and content. Hence, I am also looking into efficient traffic shaping strategies for such services.

**Memory side-channel mitigation for Cloud services.** During an internship at Microsoft Research, I looked into mitigating memory side-channel disclosures in Cloud services that rely on strong hardware-based isolation techniques (e.g., Intel SGX) to protect from even a compromised OS or hypervisor. However, despite the isolation, a compromised guest OS or hypervisor, which controls the pages tables, can observe the memory pages accessed by an application and infer its secrets. In the context of machine learning (ML) services, such side channels can leak proprietary models, and sensitive training or prediction inputs stored in memory. To plug this leak, we modified various ML algorithms (e.g., K-Means Clustering, Decision Trees, etc.) to make their memory access patterns independent of sensitive inputs (data-oblivious).

More broadly, this work presents SGX as a powerful mechanism to enable novel applications, such as collaborative ML, that require strong memory guarantees as well as efficiency. Mutually untrusting parties, such as hospitals, insurance providers, and banks, today wish to reap the benefits of rich ML models generated by collaboratively training on their private datasets. At the same time, they do not wish to share their datasets with each other, nor with a third-party. Cryptographic techniques, such as garbled circuits and homomorphic encryption, enable multiple parties to perform shared computations on encrypted data, but these techniques do not scale to complex applications involving large datasets. Resource-rich SGX enclave offerings in public Clouds, combined with mitigations for side-channel attacks can provide a trustworthy environment to run multi-party computations on private data shares even in cleartext, thus providing both security and efficiency. To realize this objective, in addition to designing memory side-channel mitigation solutions, I designed a protocol to enable parties to securely upload their private data to and retrieve results from an SGX enclave in the Cloud.

### 3 Future directions

The relevance and difficulty of ensuring compliance will only increase in the coming years as systems evolve rapidly to accommodate new hardware, programming paradigms, applications, and regulations. I am particularly excited about two emerging trends in Cloud datacenter architecture—serverless computing and hardware disaggregation—which will require new designs and mechanisms for policy compliance in future Cloud applications. Looking ahead, I am interested in addressing systems challenges involved in quickly adapting services to more and stricter regulations regarding fake news, political advertisements, hate speech, or filter bubbles. To address these new challenges, I will continue to take the compliance-by-default approach in building effective solutions.

**Compliance for next-generation Cloud applications.** With serverless computing, monolithic applications are being refactored into tens to thousands of stateless, potentially mutually untrusting functions that communicate with each other to process a single client request. Secondly, hardware vendors, such as Intel, are pushing towards disaggregated hardware

---

Aastha Mehta – Research Statement 3 of 4

[5] Pacer

[6] Underway
that replaces traditional CPUs with separate compute, memory and storage servers, which will result in further refactoring of datacenter applications. These trends lead to an increased separation between, and change in the trust relations among the software components of an application that were previously located in the same monolithic, trusted server. This brings several challenges for ensuring end-to-end compliance in an application in terms of both specifying and enforcing policies.

For instance, an application may rely on multiple stateless functions, each performing a different task and relying on different types of data stores (e.g., a relational database and a key-value store), but required to comply with a set of common data policies (e.g., a time-based policy on a user’s data items). I am interested in exploring the design of unified abstractions for data and policies that will ensure efficient as well as consistent access to the policies across different types of functions and data stores. Secondly, enforcing policies requires tracking fine-grained data flows across functions, which increases policy enforcement overheads. At the same time, the simpler design of functions also makes them more amenable to auditing and static analysis techniques, which have traditionally not scaled to large monolithic codebases. Therefore, to address the policy enforcement problem, I am interested in exploring hybrid solutions combining static analyses and runtime enforcement to enable efficient end-to-end policy compliance for applications.

Efficient mitigation of increased side-channel threats. Serverless computing and hardware disaggregation fundamentally also lead to increased, fine-grained sharing of physical resources among functions of different tenants. This makes threats of side-channel disclosures even more pertinent, as an adversary can now observe individual functions or small groups of functions, and infer information about an application’s intermediate states that was previously hidden. I would like to build solutions to mitigate such fine-grained side channels, specifically focusing on efficiency challenges.

For instance, in the context of network side channels mitigation, while Pacer’s conceptual idea of traffic shaping can be applied, its software-based implementation is unlikely to scale to the high network loads resulting from the disaggregation. I am interested in exploiting new hardware technologies, such as programmable smartNICs, to implement a secure and efficient network side-channel mitigation entirely in NIC. In addition, I would like to explore efficient techniques for mitigating other side channels (e.g., via microarchitecture). We can implement individual functions such that they execute in constant time and access the shared resources in the same order regardless of secrets. Given the simpler design of individual functions, this approach can be realized without incurring significant performance or resource overheads.

Compliance with future regulations. With the large number of privacy breaches in the recent past, Internet companies have come under the scrutiny of public as well as regulatory bodies internationally. GDPR is just the first of the regulations to be imposed on these companies today. Several other countries are formulating their own privacy regulations. In addition, many more and stricter regulations are likely to follow in the near future, which may address issues beyond user privacy, say fake news or political advertisements. For instance, the EU’s recent copyright legislation requires service providers to remove digital content violating copyrights. Quickly adapting to these new regulations across geographies has become a challenge, especially for small- and medium-scale organizations, and community-driven services (e.g., startups and discussion forums). I would like to explore how we can design compliance tools and libraries that can be adopted off the shelf, without involving significant overheads for system developers, maintainers, and operators.

In summary, my aim is to develop principles, methods, and tools that will enable service providers to build powerful, efficient, agile, and scalable systems that are compliant by default with applicable rich policies regarding data collection, dissemination, storage, and use, as well as fair and transparent information access.

References


Indian Data Protection Framework, The EU Directive on Copyright in the Digital Single Market