DuetSGX: Differential Privacy with Secure Hardware

Phillip Nguyen
Phillip.Nguyen@uvm.edu
University of Vermont

Alex Silence
Alex.Silence@uvm.edu
University of Vermont

David Darais
David.Darais@uvm.edu
University of Vermont

Joseph P. Near
jnear@uvm.edu
University of Vermont

ABSTRACT

Differential privacy offers a formal privacy guarantee for individuals, but many deployments of differentially private systems require a trusted third party (the data curator). We propose DuetSGX, a system that uses *secure hardware* (Intel's SGX) to eliminate the need for a trusted data curator. Data owners submit encrypted data that can be decrypted *only* within a secure enclave running the DuetSGX system, ensuring that sensitive data is never available to the data curator. Analysts submit queries written in the Duet language, which is specifically designed for verifying that programs satisfy differential privacy; DuetSGX uses the Duet typechecker to verify that each query satisfies differential privacy before running it. DuetSGX therefore provides the benefits of local differential privacy *and* central differential privacy simultaneously: noise is only added to final results, and there is no trusted third party. We have implemented a proof-of-concept implementation of DuetSGX and we release it as open-source.

1 INTRODUCTION

Differential privacy [14, 15] and its variants represent the "gold standard" in privacy protection, and it has been adopted by organizations like Google [16], Apple [1], and the US Census Bureau [5] to protect the privacy of individuals. However, challenges in deploying differential privacy remain.

Key among these challenges is the problem of the *trusted data curator*. In the *central model* of differential privacy [15], data owners contribute their sensitive data to a central repository maintained by a *data curator*, who runs differentially private computations on the data and reveals the results. In the central model, the data curator must be *trusted* to safeguard the sensitive data and reveal *only* differentially private results.

What if the data curator is *not trustworthy*, or becomes compromised? In that case, the adversary can simply *look at the data*, and differential privacy provides no benefit at all. To address this problem, the *local model* of differential privacy [15] requires data contributors to add noise to their data *before* submitting it to the data curator. Since this data already satisfies differential privacy when it is collected, the data curator does not need to be trusted. This approach is used in systems at Apple [1] and Google [16], due to the advantages of its threat model. Unfortunately, the local model of differential privacy requires *significantly* more noise than the central model, making the results much less accurate.

We study an important question in differential privacy research: *can we achieve the benefits of both models simultaneously?* In other words, our goal is to provide the more accurate answers that are possible in the central model, while eliminating the requirement for a trusted data curator, as in the local model.

This paper presents DuetSGX, a proof-of-concept system that uses *secure hardware* to achieve the accuracy of the central model without the requirement of a trusted data curator. DuetSGX is a platform for collecting sensitive data and performing differentially private queries on that data. The data submitted to a DuetSGX server is encrypted, and computations on the data are protected by a *secure enclave* provided by Intel's Software Guard Extensions (SGX) [12]. The secure enclave prevents other processes—including the operating system—from examining the sensitive data or intermediate results of computation. The enclave also enables *remote attestation*, a protocol by which data owners can verify that they are communicating with a valid DuetSGX server running on a real SGX processor. Data owners must trust the hardware itself (and thus its manufacturer, Intel), but do *not* need to trust the data curator in control of that hardware.

DuetSGX requires that all queries satisfy differential privacy. Raw data can never be extracted by any party from a DuetSGX server. To ensure that queries satisfy differential privacy, DuetSGX requires queries to be specified in the Duet language [22], and uses the Duet typechecker to verify that each query satisfies differential privacy and to determine its privacy cost. If the query satisfies differential privacy and its privacy cost does not exceed the remaining privacy budget, DuetSGX uses the Duet interpreter to execute the query on the encrypted sensitive data. To ensure confidentiality and integrity, DuetSGX runs the Duet typechecker and interpreter inside of the secure enclave.

We have built a proof-of-concept implementation of DuetSGX that provides a RESTful API accessible by standard HTTP requests. Our implementation re-uses the existing implementation of the Duet typechecker and interpreter, and adds capabilities for handling encrypted data and managing a global privacy budget. To run inside of a secure enclave, DuetSGX relies on Graphene [27], which provides the ability to run unmodified programs inside of enclaves. Our proof-of-concept has some important limitations (discussed in Section 5), but it demonstrates the feasibility of the approach and represents the first step towards a practical, deployable platform. Our implementation is available as open source.[1]

2 BACKGROUND

Differential Privacy. Differential privacy is a formal privacy definition that bounds the effect any single individual can have on the outcome of an analysis. Formally, we say that a *mechanism* $\mathcal{M}$ satisfies (ϵ, δ)-differential privacy if for all databases $x, y \in$

[1] https://github.com/uvm-plaid/duet-sgx

$\mathcal{D}$ with a *distance metric* $d(x, y) \in \mathbb{R}$, and for all possible sets of outcomes S, $\Pr[\mathcal{M}(x) \subseteq S] \leq e^{\epsilon}\Pr[\mathcal{M}(y) \subseteq S] + \delta$ when $d(x, y) <= 1$. We can achieve differential privacy using several basic mechanisms; for example, the *Laplace mechanism* adds noise drawn from the Laplace distribution and satisfies $(\epsilon, 0)$-differential privacy (or "pure" differential privacy), and the *Gaussian mechanism* adds Gaussian noise and satisfies (ϵ, δ)-differential privacy (or "approximate" differential privacy). Differential privacy mechanisms are *compositional*: running $\mathcal{M}$ twice on the same data satisfies $(2\epsilon, 2\delta)$-differential privacy. For a detailed summary of differential privacy, see Dwork & Roth [15].

Duet. Duet [22] is a programming language for writing differentially private programs and verifying correct use of differential privacy. In Duet, privacy is encoded through *types*, and when Duet accepts a program as well-typed, this amounts to a proof that the program will satisfy differential privacy when executed. Duet's typechecker and interpreter were previously implemented as part of the Duet project.[2] See Section 4 for more information on Duet and how it is used in DuetSGX.

Secure Hardware. Intel's Software Guard Extensions (SGX) [12] is a security feature present on most of Intel's recent CPUs. SGX is one example of a *Trusted Execution Environment (TEE)*; other examples include ARM TrustZone and AMD Secure Encrypted Virtualization. Each one provides a slightly different programming interface, but all are intended to provide *secure execution*.

SGX allows programs to launch *secure enclaves*, which protect the confidentiality and integrity of computation occurring within the enclave—even from privileged software like the operating system. To protect against untrusted parties "faking" the presence of SGX, it also provides remote attestation capabilities. Remote attestation allows the CPU to produce a *quote* certifying that (1) the software is running inside of an enclave on a legitimate SGX CPU, and (2) the software has not been tampered with. The CPU constructs the quote by computing a digest of the contents of the enclave and signing it using a private key built into the CPU itself. Intel provides an attestation service [3] for verifying the authenticity of these quotes.

3 DuetSGX Overview

This section provides an overview of the DuetSGX system: its threat model and goals (Section 3.1), its architecture and API (Section 3.2), and its implementation (Section 3.3).

3.1 Threat Model & Guarantees

The goal of DuetSGX is to ensure confidentiality for the sensitive data submitted by data owners, and to guarantee that all results computed and output by the system satisfy differential privacy. We consider a setting in which a malicious adversary may control the server running the DuetSGX platform, the network connecting that server to data owners, and the analyst who submits queries to the platform. We assume that a data owner's device cannot be corrupted by the adversary, and that the SGX hardware protections are also robust against the adversary (more on this assumption in Section 5).

[2] https://github.com/uvm-plaid/duet

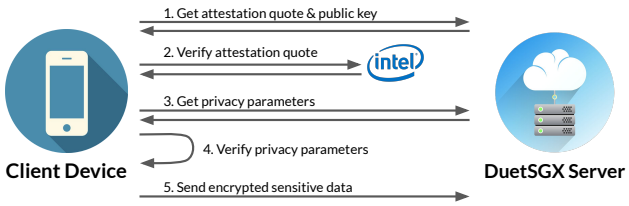

Figure 1: Data Collection Phase. In this phase, the client device negotiates with the DuetSGX server to securely submit data for differentially private analysis.

Under these assumptions, DuetSGX ensures confidentiality for submitted data, and differential privacy for all outputs. The submitted data is encrypted so that it can only be processed within a secure enclave, protecting it from the data curator. Queries submitted by the analyst are verified to satisfy differential privacy by the Duet typechecker, preventing a malicious analyst from writing a query to reveal the submitted data.

3.2 Architecture

Usage of DuetSGX proceeds in two phases: (1) data collection, and (2) query processing.

Phase 1: Data Collection. In the first phase, client-side software communicates with the DuetSGX server to contribute data. The DuetSGX server provides a RESTful HTTP interface for submitting data and queries, summarized in Figure 3. In general, a data owner submits a data element to the DuetSGX server following the process shown in Figure 1. The first step is to obtain an SGX attestation quote from the DuetSGX server and verify it with Intel's attestation service [3]. Second, the client obtains the current privacy budget settings, signed using the DuetSGX server's public key, and verifies that they satisfy the data owner's wishes. Finally, the client encrypts the sensitive data using the server's public key and submits it to the DuetSGX server.

The API is implemented using a Python Flask server and can be reached using standard HTTP requests. The DuetSGX server also provides an HTML interface using Javascript to interact with the API, allowing clients to submit data via a web browser. The web-based interface encrypts data in the browser, using a Javascript implementation of RSA.

The key pair used in this process is generated within the secure enclave when the DuetSGX server starts up, and the private key never leaves the enclave. When the server shuts down, the private key is lost, and the encrypted data that has been submitted so far is effectively destroyed.

Phase 2: Query Processing. The second phase is processing differentially private queries over the submitted data. Analysts write their queries using the Duet language (detailed in Section 4) and submit them to the `/query` endpoint of the DuetSGX server. The server runs the query and returns a differentially private result. This process is summarized in Figure 2.

The HTTP server itself is a Python program that is *untrusted*. The trusted DuetSGX binary runs entirely inside of an SGX enclave, and interacts with the untrusted HTTP server by reading and writing files on disk. As described earlier, the key pair used

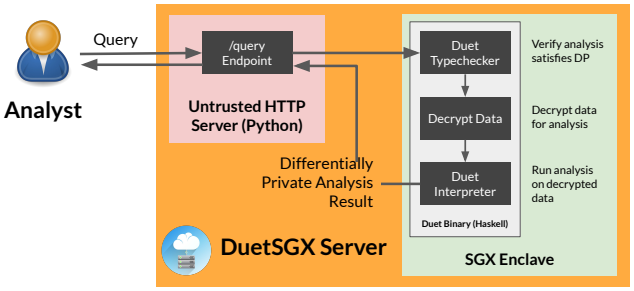

Figure 2: Data Analysis Phase. In this phase, the DuetSGX server runs Duet programs on the submitted data to compute differentially private results.

API	Method	Description
/epsilon	GET	Returns the current ϵ value
/delta	GET	Returns the current δ value
/attest	GET	Returns the attestation quote
/pubkeypem	GET	Returns the public key pem
/insert	POST	Insert data into the database
/query	POST	Submit a query for execution

Figure 3: DuetSGX Server API

for encrypting and decrypting the sensitive data is generated when the enclave starts up, and the private key is stored only in enclave memory; when the DuetSGX server shuts down, the private key is (intentionally) lost.

The privacy budget is set by the data curator when the DuetSGX server starts up by specifying the initial values for ϵ and δ in a file. When the server starts, it signs these values using the generated key pair, so that the data curator cannot modify the privacy budget after the server is running. When a query is submitted, the Duet typechecker derives a privacy cost for it and subtracts that amount from the remaining budget. If the budget would be exhausted by the query, then the query is rejected; otherwise, the query's cost is subtracted from the budget and the new value of the remaining budget is signed by the DuetSGX server.

3.3 Implementation

The DuetSGX binary that runs inside of the secure enclave is written in Haskell, and builds on the Duet implementation (which is also written in Haskell). The DuetSGX enclave code implements the specific interface with the untrusted HTTP server, handles the generation of encryption keys and encryption and decryption of data, and coordinates the use of the Duet typechecker to determine privacy cost and the Duet interpreter to compute results. The untrusted HTTP server is a simple Python program implemented using the Flask library.

Intel SGX enclaves are not normally capable of running Haskell programs, so we use the Graphene system [27] to enable DuetSGX to run on SGX. Graphene provides a compatibility layer that allows arbitrary Linux programs to run in an SGX enclave. Graphene simplifies the development of systems like DuetSGX, but also brings limitations, especially for remote attestation. We discuss these in Section 5.

4 SPECIFYING DuetSGX QUERIES

DuetSGX runs queries specified in Duet [22], a programming language and type system designed for automatically proving that a program satisfies differential privacy. When a query is submitted, the DuetSGX server runs the Duet typechecker *before* running the query, to (1) verify that the submitted query satisfies differential privacy, and (2) determine the privacy cost (ϵ, δ) of running the query.

Duet's typechecker is entirely static and does not require access to the data. Its results can therefore safely be used to determine the privacy cost of a query before running it, and to reject queries that do not satisfy differential privacy or have privacy costs greater than the remaining privacy budget.

Writing Duet Programs. Duet includes basic primitives for differentially private programming like the Laplace mechanism (written `laplace`) and the Gaussian mechanism (written `gauss`). For example, the following program adds Gaussian noise to the variable `x`, assuming that `x` has a sensitivity of 1.0:

```
let ε = ℝ⁺[1.5] in
let δ = ℝ⁺[0.000001] in
gauss[ℝ⁺[1.0], ε, δ] <x> { x }
```

The notation `ℝ⁺[1.5]` is used to specify a *statically-known* non-negative real value, which the typechecker can use to help determine privacy cost. Sensitivities and the values of privacy parameters must be statically known in order for the Duet typechecker to bound privacy cost, and statically-known values are assumed to be publicly known. The list of variables inside angle brackets (`<x>`) denote the variables for which the Gaussian mechanism should attempt to provide privacy; variables not listed here will be treated as auxiliary information and assigned infinite privacy cost. The expression inside curly braces (`{ x }`) represents the actual result to which noise will be added.

Duet also allows specifying *privacy functions* (with `pλ`) whose types encode their privacy costs. For example:

```
pλ . x : ℝ ⇒
  gauss[ℝ⁺[1.0], ℝ⁺[1.0], ℝ⁺[0.001]] <x> { x }
```

The Duet typechecker reports that this program has the type:

```
ℝ@⟨1.0, 0.001⟩ ⇒ ℝ
```

This type indicates that the program's value is a function which satisfies $(1.0, 0.001)$-differential privacy (the privacy cost is listed after the `@` sign). For more information on the Duet language, see the Duet paper [22] or the project webpage [2].

Duet Programs as DuetSGX Queries. A DuetSGX query is simply a Duet program with a specific form. A valid DuetSGX query is a program that defines a privacy function of a single argument (the data stored in the secure database). The privacy function must have constant privacy cost and the type of the function's argument must match the schema of the secure database. For example, the following program represents a DuetSGX query that counts the number of rows in a secure database of pairs of real numbers (e.g. latitude/longitude pairs):

```
pλ . df :  M [L1 , U | ★ , d ℝ :: d ℝ :: [] ] ⇒
  let ε = ℝ⁺[1.0] in
  let δ = ℝ⁺[0.001] in
  gauss[ℝ⁺[1.0], ε, δ] <df> { real (rows df) }
```

The notation M [L∞,U|...] is a Duet *matrix type*, which we use in DuetSGX queries to specify the schema of the database. The L1 annotation specifies the distance metric used to define neighboring inputs (we add up the number of differing rows), and U specifies that there is no known upper bound on values in the matrix. ★ means the number of rows in the matrix is not statically known (this will usually be the case for DuetSGX databases). d ℝ :: d ℝ :: [] specifies the schema of the matrix, in linked-list notation; this example has two columns, each containing real numbers. d ℝ is the *discrete real number* type, which is the same as a real number except with a boolean distance metric: two values $v_1, v_2 \in$ d ℝ are 0 apart if they are equal, and 1 apart otherwise. rows is a built-in function that counts the number of rows in a matrix, and real converts a discrete real number into a real number. We have used an unrealistically large value for δ in this example for readability. This program has the type:

```
M [L1,U | ★, d ℝ::d ℝ::[]]@⟨1.0, 0.001⟩ ⇒ ℝ
```

The Duet typechecker returns this type to the DuetSGX implementation, which uses the privacy cost annotation in the privacy function type to determine the privacy cost of the query and reduce the privacy budget accordingly.

Expressive Power of DuetSGX Queries. We have shown just a few simple examples of valid DuetSGX queries here, but the Duet language is designed to be flexible and support complex programs beyond simple analytics. For example, Duet supports looping constructs with differential privacy, and can use advanced composition to determine privacy cost for these programs; Duet also supports recent variants of differential privacy like Rényi differential privacy [20] and zero-concentrated differential privacy [10], and these can be used in DuetSGX queries to improve composition. In addition to programs that compute traditional analytics, we have also implemented DuetSGX queries for machine learning (e.g. noisy gradient descent algorithms [4, 6, 11]).

5 LIMITATIONS & FUTURE WORK

Our proof-of-concept implementation of DuetSGX is a prototype, and should not be used in production to protect sensitive data. It has not been tested extensively, and may contain bugs that would allow attacks that reveal the encrypted data.

In addition, limitations in Graphene result in known issues with DuetSGX that could result in security vulnerabilities. In particular, Graphene's remote attestation features are still under construction; it is not yet possible to embed DuetSGX's public key in the attestation quote that is generated. Instead, our prototype implementation exposes the public key separately, by writing it to a file. This approach means that the untrusted HTTP server could *modify* the public key before sending it to clients, which could enable an adversary who does not control the secure enclave to decrypt and read the submitted data. As Graphene's support for remote attestation improves, we plan to embed the public key in the quote itself, which is signed by the SGX processor's private key so that it cannot be modified by the untrusted components of the system.

Intel SGX has been the subject of a number of attacks [17, 28–30, 32]. Many of these attacks allow a process outside the enclave to recover secrets stored inside the enclave memory, including private keys, and would enable an attacker to defeat the protections of DuetSGX. Most of the existing attacks have been addressed quickly by Intel, and can be mitigated by updating the processor's microcode, but this issue should be carefully considered before SGX is used in any system to protect truly sensitive data. We plan to adapt our implementation of DuetSGX to new implementations of secure enclaves as they appear—AMD's Secure Encrypted Virtualization, for example, will allow running unmodified programs—and we expect fewer realistic attacks as secure hardware matures.

Finally, our implementation of DuetSGX does not support the complete range of features of the underlying Duet system. In particular, the privacy budget for DuetSGX is stored in terms of values for ϵ and δ; programs that leverage other variants of differential privacy (e.g. Rényi differential privacy or zero-concentrated differential privacy) can be written as DuetSGX queries, but their privacy costs must be converted to (ϵ, δ)—DuetSGX is not currently capable of using these variants for sequential composition of queries. We plan to explore additional features for composition—including variants like these and also the ability to submit workloads of queries for improved accuracy.

6 RELATED WORK

The use of SGX to provide security for outsourced computation has been studied extensively, and systems like Opaque [33], VC3 [26], Haven [8], Ironclad Apps [18], and Ryoan [19] achieve this goal in various ways. These approaches typically assume the analyst to be trusted, and do not integrate differential privacy.

Ongoing research in the local model of differential privacy has produced a significant number of systems, including RAPPOR [16], Prochlo [9], Apple's system [1], and many other works in various kinds of analytics [23, 31] and machine learning [13]. Recent work has explored combining differential privacy with cryptographic techniques for secure computation: Honeycrisp [24] uses a secure aggregation protocol to eliminate the trusted data curator for various analytics, and DJoin [21], Shrinkwrap [7], and Cryptϵ [25] run a variety differentially private analytics queries using multiparty computation at a smaller scale. These systems do not require special hardware, but they introduce significant performance overhead and provide a less expressive query language than DuetSGX.

7 CONCLUSION

We have presented DuetSGX, a platform for collecting and processing sensitive data that uses secure hardware to guarantee confidentiality for the data and ensure differential privacy for all outputs. DuetSGX collects and stores encrypted data, and allows its decryption only within a secure enclave that executes *only* differentially private queries. Queries are written using the Duet language and verified to be differentially private using the Duet typechecker. Our proof-of-concept implementation demonstrates that secure hardware can be used to build systems that provide the benefits of both the local and central models of differential privacy, without requiring a trusted data curator.

ACKNOWLEDGMENTS

We would like to thank the TPDP20 reviewers for their helpful suggestions for improvements. This work was supported by NSF via award CCF-1901278, by ODNI/IARPA via award 2019-1902070008, and by DARPA & SPAWAR under contract N66001-15-C-4066. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] Apple: Differential privacy overview. https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf.

[2] Duet project homepage. https://github.com/uvm-plaid/duet.

[3] Intel attestation service. https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions/attestation-services.html.

[4] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[5] John M Abowd. The us census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2867–2867, 2018.

[6] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 464–473. IEEE, 2014.

[7] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. Shrinkwrap: efficient sql query processing in differentially private data federations. *Proceedings of the VLDB Endowment*, 12(3):307–320, 2018.

[8] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS)*, 33(3):1–26, 2015.

[9] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 441–459, 2017.

[10] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*, pages 635–658. Springer, 2016.

[11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(3), 2011.

[12] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[13] John C Duchi, Michael I Jordan, and Martin J Wainwright. Local privacy and statistical minimax rates. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 429–438. IEEE, 2013.

[14] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[15] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[16] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[17] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, pages 1–6, 2017.

[18] Chris Hawblitzel, Jon Howell, Jacob R Lorch, Arjun Narayan, Bryan Parno, Danfeng Zhang, and Brian Zill. Ironclad apps: End-to-end security via automated full-system verification. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 165–181, 2014.

[19] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM Transactions on Computer Systems (TOCS)*, 35(4):1–32, 2018.

[20] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275. IEEE, 2017.

[21] Arjun Narayan and Andreas Haeberlen. Djoin: Differentially private join queries over distributed databases. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 149–162, 2012.

[22] Joseph P Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, et al. Duet: an expressive higher-order language and linear type system for statically enforcing differential privacy. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–30, 2019.

[23] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. Heavy hitter estimation over set-valued data with local differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 192–203, 2016.

[24] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: large-scale differentially private aggregation without a trusted core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 196–210, 2019.

[25] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. Cryptε: Crypto-assisted differential privacy on untrusted servers. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 603–619, 2020.

[26] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. Vc3: Trustworthy data analytics in the cloud using sgx. In *2015 IEEE Symposium on Security and Privacy*, pages 38–54. IEEE, 2015.

[27] Chia-Che Tsai, Donald E Porter, and Mona Vij. Graphene-sgx: A practical library {OS} for unmodified applications on {SGX}. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pages 645–658, 2017.

[28] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 991–1008, 2018.

[29] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAxe: How SGX fails in practice. https://sgaxeattack.com/, 2020.

[30] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. Cacheout: Leaking data on intel cpus via cache evictions, 2020.

[31] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. Locally differentially private protocols for frequency estimation. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 729–745, 2017.

[32] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves. In *European Symposium on Research in Computer Security*, pages 440–457. Springer, 2016.

[33] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 283–298, 2017.