Quantitative and Qualitative Investigations into Trusted Execution Environments

Ryan Karl(✉)

University of Notre Dame, Notre Dame, IN 46556, USA
rkarl@nd.edu

Abstract. I propose to develop a quantitative and qualitative framework to integrate a Trusted Execution Environment (TEE) into the pipeline of secure computation by combining it with other cryptographic primitives. Such a hybrid framework will utilize mathematical and statistical modeling techniques to decide how to combine TEE and cryptographic primitives and evaluate the potential for performance improvement by moving secure computation processes into or out of a TEE. Ideally, I will be able to determine when to combine TEEs with pure-cryptography techniques to improve performance for a task, instead of simply using either method alone and only achieving suboptimal performance. The final goal is to design and develop an actionable decision-making framework, and utilize it to optimize the secure computation process.

Keywords: Trusted Execution Environment · Hybrid secure computation · Performance optimization and modeling

1 Introduction

Background: Third-party analysis on private records is becoming more important due to widespread data collection for various analysis purposes in business, government, academia, etc. This can be observed in many real life applications, such as the Smart Grid, Social Network Services, Location Based Services, etc.[8]. While existing service providers support large scale high performance computation over unencrypted data, data in its plaintext form often contains private information about individuals, and the publication of such data may violate legal agreements such as HIPPA, GDPR, etc. Current best practices for secure, privacy preserving computation over this data (i.e., Secure Multiparty Computation (MPC), Fully Homomorphic Encryption (FHE), etc.) generally incur huge computation and/or communication overheads when compared to computing over plaintext data. A promising approach towards secure computation, that can be significantly faster than expensive cryptographic approaches, is to utilize a Trusted Execution Environment (TEE), a secure area of the main processor

J. Garcia-Alfaro et al. (Eds.): SecureComm 2021, LNICST 399, pp. 372–383, 2021.
https://doi.org/10.1007/978-3-030-90022-9_19

that leverages hardware and software components to guarantee code and data loaded inside the TEE will be protected from malicious tampering or eavesdropping by software operating at a higher privilege level.

Research Goal: While TEEs are a promising technology that have seen major adoption in industry, their performance has not been rigorously studied in comparison to pure cryptography techniques. Currently there is no comprehensive, up-to-date study that provides qualitative and quantitative information to prospective users on whether a TEE is a useful addition to their system. The general assumption is that TEE is always superior to cryptographic primitives, however there exist lightweight primitives that have high efficiency and/or throughput. In order to better understand the benefits of deploying TEEs in the wild, I propose to design a framework that can quantify the potential for performance improvement or degradation by migrating all or part of a system into or out of a TEE.

The proposed framework will ensure that users have a deep understanding of what segments of code are the best candidates for a migration to a TEE to improve performance. Note that my framework will not ignore existing data protection mechanisms (e.g., pure cryptography solutions). Rather, it tries to complement the competing privacy-preserving computation methods, to better understand the advantages and drawbacks of TEEs when compared to more traditional techniques. Ideally, it will identify situations where using both techniques together will offer superior performance to using one of the techniques in isolation. The contribution of the proposed framework is noteworthy: it assists security specialists in analyzing their projects and determining if a TEE is right for them based on both performance and resource availability. It will contribute to the adoption and growth of the intelligent use of TEEs in the data privacy ecosystem and protect potential users from naively using TEEs incorrectly and slowing down their system performance.

Research Challenges and Intellectual Merit: Many challenges exist in the framework design. TEEs come with their own set of constraints, including computational overhead due to expensive encrypted context switching operations, a lack of access to reliable and sophisticated parallel computing technologies, and code size constraints. Therefore, it is challenging to know which tasks are best suited for migration into a TEE, and which are better suited for other pure cryptography techniques. A rigorous analysis would be of great value to system designers during the early stages of planning their system designs, so they could avoid investing too much time in a technology that is not well suited to their application. No one has yet attempted a quantitative study to model which types of computing paradigms or operations are best suited for either domain, which adds difficulty to this task.

Preliminary Work: My own prior work suggests that there are several scenarios where combining TEEs and pure cryptographic approaches together can provide superior performance to simply using one technique on its own. Many protocols are complex enough that the logic of certain components are better suited for TEEs, while other components are more ideal for utilizing pure cryptography techniques, and this is discussed in more detail in Sect. 3. No one has yet

attempted a quantitative study to model which types of computing paradigms or operations are best suited for either domain or how to develop a hybrid computing approach that combines pure cryptography with TEEs to obtain optimal performance. These challenges add intellectual merit to this task.

Broader Impacts: In the long term, the proposed project has the potential to assist security specialists in analyzing their projects and quantifying how a TEE can improve their system's performance based on their resources. This project's broader impact will contribute to the adoption and growth of the use of TEEs in the data privacy ecosystem, and protect potential users from incorrectly using TEEs in a way that introduces unacceptable vulnerabilities or slows down their systems. I propose to utilize theoretic techniques to model runtime, and also run empirical tests to determine the real world runtime of computational tasks. This will determine which tasks are better suited for TEEs and which are more appropriate to pure cryptography techniques. I plan to leverage the framework to inform the system design process when investigating whether to migrate part of a real world secure polynomial aggregation protocol into or out of a TEE. Each of the methodologies and TEEs discussed in this framework will be evaluated thoroughly to determine the pros and cons of using a TEE to solve this research problem.

2 Privacy-Preserving Computation Today

Although there is also a growing need for technology that supports privacy-preserving time-series data analysis, there are relatively few techniques that offer practical levels of performance and accuracy. Fully Homomorphic Encryption (FHE), Differential Privacy (DP), Secure Multiparty Computation (MPC), or Oblivious Polynomial Evaluation (OPE) might be used individually as black boxes to solve this problem, but each have significant constraints that negatively impact their practical deployment in the real world [9]. FHE's high computational overhead leads to significant slowdown that makes it impractical in large-scale settings. DP adds noise to the final output of the function, and the resulting accuracy loss can greatly harm the predictive power of any analysis. MPC requires participants to send multiple messages during protocol execution, which can seriously degrade overall runtime. OPE also requires multiple messages to be sent, and is primarily focused on the two party setting, which limits its applicability in large scale data analysis.

Private Stream Aggregation (PSA): PSA is a form of distributed secure computing that is promising for achieving this functionality. With this technique, users independently encrypt their input data and send it to an aggregator in a way that allows the aggregator to efficiently learn the aggregation results of time-series data without being able to infer individual data. PSA is generally superior to other types of secure computation paradigms (e.g., MPC, FHE) in large-scale applications involving time-series data because of its extremely low overhead and the ease of key management [11]. Notably, PSA is non-interactive (i.e., users send their time-series data in a "stream" and only one message is

sent per time interval) and asynchronous (i.e., users can leave after submitting their inputs), making it more efficient in communication than most existing alternative techniques [14]. Although PSA is a mature field of study, prior work in this field is mostly limited to simple aggregation (sum, average, etc.). Due to these limitations, it is challenging for even the most advanced PSA protocols to be deployed in real-world applications.

Trusted Execution Environments (TEEs): One of the most prevalent TEEs in modern computing is Intel SGX. I review Intel SGX below, as most modern TEEs support a majority of the services it provides. Intel SGX is a set of new CPU instructions that can be used by applications to set aside private regions of code and data. It allows developers to (among other things) protect sensitive data from unauthorized access or modification by malicious software that may be running at superior privilege levels. To do this, the CPU protects an isolated region of memory called Processor Reserved Memory (PRM) against other non-enclave memory accesses, including the kernel, hypervisor, etc. Sensitive code and data is encrypted and stored as 4KB pages in the Enclave Page Cache (EPC), a region inside the PRM. Even though EPC pages are allocated and mapped to frames by the OS kernel, page-level encryption guarantees confidentiality and integrity. In addition, to provide access protection to the EPC pages, the CPU maintains an Enclave Page Cache Map (EPCM) that stores security attributes and metadata associated with EPC pages. This allows for strong privacy and integrity guarantees if applications can be written in a two part model [4].

Applications must be split into a secure part and a non-secure part. The application can then launch an enclave, which is placed in protected memory, that allows user-level code to define private segments of memory, whose contents are protected and unable to be read or saved by any process outside the enclave. Enclave entry points are defined during compilation. The secure execution environment is part of the host process, and the application contains its own code, data, and the enclave, but the enclave contains its own code and data too. An enclave can access its application's memory, but not vice versa, due to a combination of software and hardware cryptographic primitives. Only the code within the enclave can access its data, and external accesses are always denied. When it returns, enclave data stays in the protected memory. In some operating systems, enclaves must be less than 128 MB, which presents a constraint on the size of SGX dependent programs. The enclave is decrypted "on the fly" only within the CPU itself, and only for code and data running from within the enclave itself. This is supported by an autonomous piece of hardware called the Memory Encryption Engine (MEE) that protects the confidentiality and integrity of the CPU-DRAM traffic over a specified memory range. The code running within the enclave is therefore protected from being "spied on" by other code. Although the enclave is trusted, no process outside it needs to be trusted (including the operating system itself) [4]. Before performing computation on a remote platform, a user can verify the authenticity of the trusted environment. By using the attestation mechanism, a third party can establish that software is running on an Intel SGX enabled device and within an enclave.

3 Open Questions to Answer

Although a large variety of techniques for secure computation exist, currently two of the most practical (in terms of performance) are lattice-based Private Stream Aggregation and Trusted Execution Environments. However, although it is known these techniques can be utilized to efficiently process different types of tasks, there is still a great deal of uncertainty regarding which techniques are best suited for different types of computation. I seek to answer the following research questions: "How much speedup, if any, can I expect to gain if I migrate sections of code into a TEE? Also, can I meaningfully quantify any potential speedup while taking resource costs into account? Furthermore, can I organize computational tasks in a meaningful way that will assist potential TEE users in designing their computation pipeline?"

Advantages of TEEs. A significant advantage of TEEs over PSA is support for branching computer logic (i.e., if else statements). Modern TEEs allow users to simply utilize basic C++ syntax to support such statements. However, due to the underlying mathematics, PSA does not have any built in techniques for achieving this functionality, since conditional statements on ciphertexts cannot be evaluated (practically), because the encrypted values cannot be seen to determine which branch to take. There is some work that leverages Lagrange Interpolation [12] to formulate a polynomial function that allows for homomorphic thresholding, but this techniques requires an expensive preprocessing step that makes it far more impractical for computing conditional statements than utilizing a TEE. In addition, TEEs do not suffer from noise when performing addition or multiplication over encrypted values. This means we can perform as many computations over encrypted data as we would like without encountering any slow down. With lattice-based PSA, every operation adds some noise to the result of the computation, such that in practice there is a limit to the amount of computation that can be performed before the result is not recoverable. Note that it is possible to utilize a technique known as bootstrapping to remove this noise periodically, in order to allow for an unlimited number of operations to be computed over ciphertexts, but this technique adds significant overhead to the overall runtime of a program, and in practice it is usually not used.

Advantages of PSA. TEEs do have several drawbacks that raise questions regarding when they are superior to or inferior to PSA. By their nature, TEEs require that users have specialized trusted hardware and firmware in order to utilize the technology, and this raises some compatibility issues in certain environments that can prevent TEE code from being as portable as PSA based techniques. Also, current TEEs are limited to a finite amount of space that can be set aside in a system to store encrypted data. This space is typically known as an enclave, and is generally limited to 128 or 256 MB on state of the art TEEs. In practice, TEEs can encounter significant overhead when paging is triggered, due to overflowing the space constraints of an enclave, and this makes computing over large quantities of data impractical when compared to PSA. However, Intel recently announced that it plans to dramatically increase the maximum size of enclaves in future releases to be approximately 1 TB, so it is unclear if

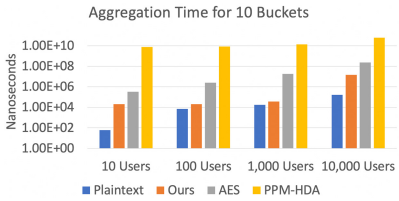


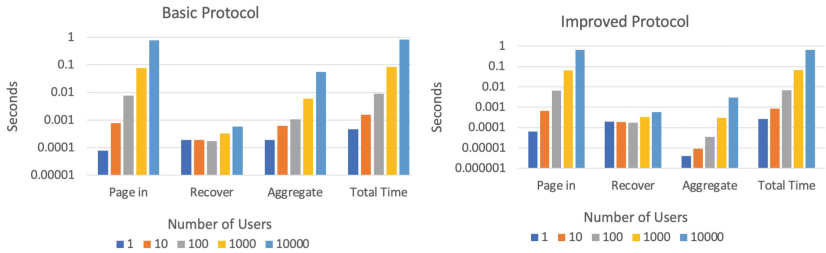
Fig. 1. Histogram aggregation experimental results

this issue will remain significant in the long term [1]. In addition, current TEEs have difficulties utilizing parallel computing techniques, such as multi-threading [13] due to the lack of common synchronization primitive support often found on traditional operating systems. Also, leveraging threading within the Intel SGX can introduce security vulnerabilities [15]. Although efforts are underway to provide additional support for parallel computing, in order to mitigate against side-channel attacks (particularly timing attacks) it seems that for the immediate future parallelism inside a TEE will be inferior to standard techniques available from an OS.

4 Current Stage of Research

During previous projects, I did some exploratory work investigating the usefulness of combining TEEs with PSA to improve performance. The results suggest that for different problems, each method may sometimes be superior if used on its own, but there are circumstances when combining the two approaches may ultimately yield better performance. For instance, when I investigated methods to improve the speed of private histogram calculations, I found that utilizing standard encryption algorithms, such as AES, to simply pass data into an enclave to be computed over, or pure cryptography based solutions (such as the PPM-HDA [7]), that does not utilize a TEE, does not always offer the best performance overall. Instead, our novel approach of leveraging a TEE combined with shuffling algorithms outperformed these baseline techniques, and our approach was within one order of magnitude of plaintext computations in some scenarios, as seen in Fig. 1. This suggests that intelligently combining PSA with a TEE may at times offer the best overall performance potential.

I also observed this when investigating techniques to provide non-interactive fault tolerance to PSA schemes by leveraging a TEE. In the original design, I incurred significant additional computational overhead, since the aggregation step was done inside the TEE. In an effort to improve efficiency, I outsourced some aggregation computation to the untrusted aggregator to improve performance and avoid the MEE's overhead. Current literature suggests most TEEs run common functionalities over an order of magnitude slower than what can be achieved on comparable untrusted hardware, due to the overhead of computing within the enclave [10]. This protocol modification allowed for the somewhat expensive aggregation step to be done on more powerful, albeit untrusted



(a) Overall Time of Basic Scheme (b) Overall Time of Improved Scheme

Fig. 2. Cryptonite (PSA) experimental results

hardware, that has better access to parallel computing resources, without compromising security. This effect can be observed in Fig. 2, where the aggregation time can be dramatically improved if this is performed outside the TEE. This suggests that combining TEEs with pure cryptography can lead to performance improvement over simply leveraging TEEs on their own.

Based on these observations, I conjectured that their may be instances where a hybrid approach, that combines TEEs with pure cryptography techniques, such as PSA, may offer ideal performance that avoids the trade offs of using either technique in isolation. With this in mind, I designed a framework to support privacy-preserving polynomial aggregation, by combining any preexisting PSA scheme with a TEE. Note that PSA based on pure cryptography only supports either addition or multiplication, but I need to leverage both to compute a polynomial function. This seems to suggest that traditional PSA alone cannot support polynomial calculations, but integrating a TEE allows additional secure functionality that supports computing more complex functions. This technique essentially performs the multiplicative components of polynomial evaluation using PSA, but then decrypts the intermediate data inside the enclave, so that privacy is still preserved, while computing the additions needed to recover the final output. During a case study when I simulated computing a regression task over several public datasets from the UCI database, I compared the performance of this method of combining a TEE with PSA with an approach that only utilizes pure cryptography (PDA [8]). I found that this method is at least an order of magnitude faster, as shown in Fig. 1. This result is encouraging, and I feel that a deeper investigation of which components should be done in a TEE and which should be outsourced is needed to obtain optimal overall performance. This problem may be ideal to use as a case study for validating our framework.

5 Proposed Methodology

5.1 Framework Design

I plan to quantitatively examine TEE performance when compared to existing secure computation techniques (i.e., PSA) to determine which types of program

Table 1. Cryptonomial performance on UCI dataset

Datasets	Records	Features	Our time	PDA time	Speedup
Census	48,842	14	2.85 s	355 s	125×
Bank	45,211	17	2.75 s	341 s	124×
Insurance	9,822	14	0.64 s	74 s	115×
White wine	4,898	11	0.35 s	33 s	94×
Red wine	1,599	11	0.17 s	12 s	71×

logic are better suited for TEEs, and which might be faster outside a TEE using advanced parallel computing over encrypted data. Through the course of this investigation, I plan to develop a mathematical/statistical model that can be used to investigate program logic that computes over plaintexts, and determine which approach will offer better performance, or if there might be merit in constructing a hybrid approach that uses both techniques. Although I may change the modeling techniques I use during the investigation based on our results, currently regression analysis [5] seems to be the most promising technique for this study. Regression analysis is frequently used successfully for prediction and forecasting after training a model using numerical data [5]. Also, it is known that regression analysis can be used to infer causal relationships between independent and dependent numerical data [2]. As a result, it seems to be a strong candidate method for developing a general model to better understand how to estimate overall runtime based on experimental data. However, regression analysis is known to have several weaknesses that may require us to utilize different techniques [5]. I plan to develop benchmarks that evaluate the runtime and algorithmic complexity for the following operations, to determine which are faster for each approach (Table 1):

1. Integer add/subtract/multiply/divide
2. Float add/subtract/multiply/divide
3. Trigonometric Functions (sine, cosine, ..)
4. Variable Declaration
5. Assignment Operation
6. Logical Operations (i.e., $\geq, \leq$, etc.)
7. 1D array allocation
8. Vector addition/subtraction/product (i.e. dot, cross. etc.)
9. Matrix addition/subtraction/product (i.e. dot, cross. etc.)
10. Other Relevant Operations

In addition, I will need to investigate the impact of parallel computing when performing this profiling, since although TEEs are limited to CPUs at this time, purely cryptographic techniques can take advantage of advanced hardware resources such as GPUs, etc., that will impact the overall performance potential. With enough empirical data, I should be able to apply a number of techniques, such as regression analysis, to fit the data and learn in detail what trends occur

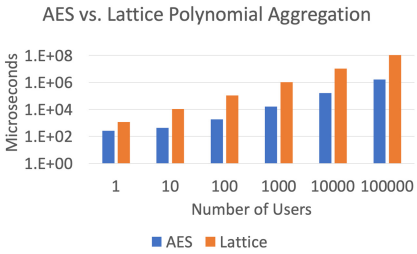


Fig. 3. Lattice aggregation vs. TEE aggregation

as I increase the amount of data and operations performed. While this may seem simple, there are a number of complicating factors that often make it unclear which technique offers faster performance/speed.

If I perform most computation in the TEE, I seem to gain processing time improvement over PSA based approaches, since AES encryption and decryption are significantly faster than their lattice based counterparts, due to a variety of factors, including hardware optimization. Also, it is known that computing basic operations over homomorphically encrypted data is considerably slower than performing the same operations over their associated plaintexts. However, the context switch involved in passing data between trusted and untrusted space is known to be a significant bottleneck in many applications, and if I exceed the size of the enclave, this will induce severe paging overheads that have the potential to significantly decrease performance. Also, it is known that some TEEs (e.g., Intel SGX) have difficulties exploiting multi-threading [13] due to the lack of common synchronization primitives often found on traditional operating systems, and leveraging threading within TEEs can introduce security vulnerabilities [15] which compromise data privacy.

On the other hand, although AES encryption is faster than FHE based techniques, recently there have been a number of significant optimizations proposed for lattice-based PSA that significantly improve its performance for certain scenarios. Besides increasing the overall runtime, these optimizations allow us to use the SIMD paradigm when using lattice based PSA, that can dramatically improve runtime for many common applications. This combined with advanced parallel computing techniques on high performance hardware, such as GPUs, could provide better performance in certain specific scenarios. Due to this large variety of complicating factors, it is critical to obtain foundational data to learn which approach offers better performance in specific scenarios. After gathering the data, I will be able to build a theoretical model to understand which logic is impacted by bottlenecks in either approach, build analytical tools to help guide development processes.

Our preliminary data, shown in Fig. 3 suggests that without utilizing lattice-based optimizations such as SIMD, and without utilizing side-channel attack resistant logic inside the TEE, utilizing trusted hardware can be significantly faster than a pure cryptography approach. To have a more nuanced understanding of the

differences in performance between the two approaches, I must first develop program logic for inside the TEE that utilizes mitigation strategies to protect against side channel attacks [6]. Then, I must develop program logic using lattice-based cryptography with optimizations (i.e., RNS, SIMD, etc.) and parallel computing techniques. Note that because I can encode multiple plaintext values into a single ciphertext using SIMD optimization, it is nontrivial to adapt some algorithms to take full advantage of the SIMD's potential to improve performance.

5.2 Case Study

The final remaining problem that I plan to address is determining if there are ways to improve the performance of existing protocols for privately computing a polynomial function over a group of users' inputs. More specifically, I seek to answer the question "Is it possible to improve the current performance of privacy preserving polynomial stream aggregation algorithms by intelligently migrating subsections of the algorithms' implementation into or out of a TEE?" Specifically, I considered the problem of allowing a set of users in S to privately compute a polynomial function over their collected time-series data, such that an untrusted aggregator only learns the final result, and no individual honest user's data is revealed. More formally, I aim at supporting polynomial evaluation over users' time-series input data in the following format of a *general multivariate polynomial*: $f(\{x_{i,j}\}_{i\in S, j=1,\ldots,z}) = \sum_{j=1}^{z} c_j(\prod_{i\in S} \mathbf{m}_{i,j,ts}^{e_{i,j}})$, where z is the number of product terms in the polynomial, c_j and $e_{i,j}$ are public parameters, and $\mathbf{m}_{i,j,ts}$ are secret data from the i-th user at time stamp ts.

Previously, I developed a framework, which can convert any PSA scheme amenable to a complex canonical embedding [3] into a privacy-preserving stream polynomial evaluation scheme, that supports general stream polynomial evaluation. Currently, The framework performs the majority of computation outside of the TEE, and only a small amount inside the TEE. However, during the design and implementation of the protocol, I speculated that our framework was flexible enough that it might be possible to modify it to either perform additional computation inside of the TEE, to potentially improve performance, or move more computation outside of the TEE , to lessen the reliance on trusted hardware and improve performance.

The first path forward would be to potentially pass everything into the enclave to speed up computation, by cutting down on the amount of comparatively expensive homomorphic operations. Instead, I might have all parties agree on a polynomial function to compute, encrypt all of their data with a traditional cryptographic primitive, such as AES, and send it into the enclave to be decrypted and aggregated. Alternatively, to compute the polynomial, I might be able to add in multiplicative and/or additive secret shares, that I could perhaps use to mask the final output until the sum of all of the products are combined.

The approach to do most computation in the TEE seems promising, since AES encryption and decryption are significantly faster than their lattice based counterparts, due to a variety of factors. In addition, there is the potential that I might have a much smaller amount of data to pass into the enclave, and the

context switch involved in passing data between trusted and untrusted space is known to be a significant bottleneck in many applications. Note that the AES ciphertexts for each user would likely be only roughly 16 bytes in size, whereas their lattice-based counterparts can be roughly 10 KB in size.

However, there are cons involved too that also make the pure cryptography approach promising as well. It is known that some TEEs (e.g., Intel SGX) have difficulties exploiting multi-threading [13] due to the lack of common synchronization primitives often found on traditional operating systems, and leveraging threading within TEEs can introduce security vulnerabilities [15] which compromise data privacy. Further, the MEE adds additional overhead into computing over data inside the TEE, and there are also space constraints. Nevertheless, computing this result without a TEE might have better performance, as I would have access to the full amenities of modern parallel computing, including special high performance hardware, such as GPUs, etc. My hope is that by performing a deeper quantitative and qualitative investigation into TEEs, it will become apparent which pipeline offers the best performance for this research problem.

References

1. Bradley, T.: Intel takes confidential computing to another level with 'ice lake' security capabilities. Forbes (2020)
2. Chatterjee, S., Hadi, A.S.: Regression Analysis by Example. John Wiley & Sons, Hoboken (2015)
3. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Floating-point homomorphic encryption. IACR Cryptol. ePrint Arch. **2016**, 421 (2016)
4. Costan, V., Devadas, S.: Intel SGX explained. IACR Cryptol. ePrint Arch. **2016**, 86 (2016)
5. Draper, N.R., Smith, H.: Applied Regression Analysis, vol. 326. John Wiley & Sons, Hoboken (1998)
6. Götzfried, J., Eckert, M., Schinzel, S., Müller, T.: Cache attacks on intel SGX. In: Proceedings of the 10th European Workshop on Systems Security, pp. 1–6 (2017)
7. Han, S., Zhao, S., Li, Q., Ju, C.H., Zhou, W.: PPM-HDA: privacy-preserving and multifunctional health data aggregation with fault tolerance. TIFS **11**(9), 1940–1955 (2015)
8. Jung, T., Han, J., Li, X.Y.: PDA: semantically secure time-series data analytics with dynamic user groups. TDSC **15**(2), 260–274 (2018)
9. Liu, D., Yan, Z., Ding, W., Atiquzzaman, M.: A survey on secure data analytics in edge computing. IEEE Internet Things J. **6**(3), 4946–4967 (2019)
10. Mofrad, S., Zhang, F., Lu, S., Shi, W.: A comparison study of intel SGX and AMD memory encryption technology. In: ACM HASP, pp. 1–8 (2018)
11. Shi, E., Chan, H., Rieffel, E., Chow, R., Song, D.: Privacy-preserving aggregation of time-series data. In: NDSS. Internet Society (2011)
12. Tan, B.H.M., et al.: Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. In: IEEE TDSC (2020)
13. Tramer, F., Boneh, D.: Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In: ICLR (2018)

14. Valovich, F., Aldà, F.: Computational differential privacy from lattice-based cryptography. In: Kaczorowski, J., Pieprzyk, J., Pomykała, J. (eds.) NuTMiC 2017. LNCS, vol. 10737, pp. 121–141. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76620-1_8
15. Weichbrodt, N., Kurmus, A., Pietzuch, P., Kapitza, R.: AsyncShock: exploiting synchronisation bugs in intel SGX enclaves. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9878, pp. 440–457. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45744-4_22