Towards Leakage-Resistant Machine Learning in Trusted Execution Environments

Abstract

Emergence of Trusted Execution Environments (TEEs), e.g., SGX and ARM TrustZone in cloud services has led to the processing of many security critical data, e.g., medical records, financial records, etc., in the cloud. However, TEEs are known to be vulnerable to side channel attacks, in which the untrusted operating system infers secret data by observing the behaviour of an enclave during its execution. One possible remedy for this problem is to write information flow secure, i.e., no secret-dependent (constant-time) branch code.

In this ongoing work, we develop a formally verified information flow secure (constant-time) gradient descent algorithm, except we instantiate the gradient calculation with (axiomatic differential private) linear regression. Moreover, we demonstrate the usability of our application by running it on synthetically generated data and the results are promising.

Keywords: formal verification, information flow security, machine learning

ACM Reference Format:

. . Towards Leakage-Resistant Machine Learning in Trusted Execution Environments. In *Program Analysis and Verification on Trusted Platforms (PAVeTrust) Workshop*. ACM, New York, NY, USA, 3 pages.

1 Introduction

In recent years, many organisations are moving towards machine learning to get the maximum utility from their data, generated by the various processes. However, machine learning algorithms are computationally expensive, many organisations are outsourcing the training to cloud (service providers). In most cases, training machine learning models in cloud is not an issue but in some situation when data contains sensitive information, e.g., health care, political view, sexual orientation, etc., cloud providers can't be trusted with processing sensitive data. Trustworthy Execution Environment, such as Intel SGX [15] and ARM TrustZone [3], has emerged as a promising solution, to execute a program on sensitive data in a untrustworthy cloud machine.

, ,

Trusted Execution Environment (SGX) guarantees isolated environment, also known as enclave, to protect the code and data from the outside software environment, including the operating system. Therefore, it has found numerous usage in security critical application, including machine learning on sensitive data [11–13, 16, 22]. Although SGX thread model provides security guarantee from many attacks (cite the paper of security guarantee); it does not promise any security from information flow attacks [4, 10, 14, 17–19, 23–25]. However, to mitigate these side channels, Intel has suggested three "constant time" principals[1]:

- Ensure runtime is independent of secret values.
- Ensure code access patterns are independent of secret values.
- Ensure data access patterns are independent of secret values.

In addition, Intel has stated that:

> These principles are conceptually simple, but sometimes can be difficult to implement in practice, depending on the complexity of your algorithms. Luckily, most developers won't need to work on the most complex situations.

Indeed, these principals are difficult to implement in complex algorithm because [11, 13, 22] are susceptible to these attacks. There are some proposals [5, 20, 21] to mitigate the side channel attacks; however, in presence of Frontal [19] attack and CopyCat [18] attack, a true solution, we believe, is formally ensuring that the program, executing in SGX, is constant time, independent of any secret branching or memory access.

In our ongoing work, we report first, to the best of our knowledge, formally verified constant time gradient descent algorithm [1], where the gradient computation is instantiated with differential private linear regression [2]. We have used the tool SecCSL [9] to develop and prove that our implementation is memory safe and ensures all the above three principles, suggested by Intel. Our experiments shows promising results on synthetic data, generated by some predefined equation of a line. In future work, we intend to extend our formally verified C code to run inside SGX, running experiments on real world data, and making it more amenable to federated learning.

[1] https://software.intel.com/content/www/us/en/develop/articles/software-security-guidance/secure-coding/mitigate-timing-side-channel-crypto-implementation.html

2 Technical Details

At a very high level, our algorithm (Algorithm 1) is very similar to [1], except we instantiate the gradient calculation by linear regression because our goal, very similar to [2], is to process relatively small amount of sensitive data [6–8], resulting from social science experiments. However, on the contrary to [2], we are mostly interested in running the (data) analysis securely in TEEs without any side channel information leak. Therefore, we take an axiomatic approach for differential privacy, i.e., we assume that the Laplacian noise generator unverified (library) function is correct, right amount of noise is added during the analysis, etc. However, to compensate these assumption, we generate certificate that records various crucial information, during the execution of program. In case of discrepancy, an auditor can inspect the certificate and ascertain the validity of execution.

Now, zooming to the details, (Algorithm 1) takes input as number of data points (n), number of iterations (T), clipping range(t), initial guess of two parameters (θ_1^0, θ_2^0), learning rate (γ), and privacy parameter (ϵ) and in each iteration, it performs:

- for every x_i, in the data:
  - it computes/predicts $\tilde{y}_i$, using the (so far) computed θ_1^t and θ_2^t (line 5).
  - then it computes the gradient $\Delta_{i,t}$, using x_i, y_i, and $\tilde{y}_i$ (line 6).
  - finally, the gradient $\Delta_{i,t}$ is clipped, i.e., if the value $\Delta_{i,t}$ is below $-t$ then it is changed to $-t$, if it is above t then it is changed to t, otherwise it remains unchanged (line 7).
- we sum all the (clipped) gradients and add Laplacian noise (line 9)
- and finally, compute (improve) new values θ_1^{t+1}, θ_2^{t+1} (line 10 and it is known as taking a descent).

We use *Secure C* [9] to formally verify that our implementation is constant-time. During our modelling, we assume that the training data is secret and therefore we assign it a *high* value. What it means is that we can no longer branch or loop on the training data or any other derived value from it, to avoid the secret-dependent branching. However, there are some steps, e.g., gradient clipping (line 7) which can be naturally expressed using *if else* construct, but we cannot because of no branching on secret rule. Therefore, we write it without *if else* construct and prove that it is correct (see the snippet, Gradient clipping snippet 1, from the gradient clipping function).

3 Experiments

Our experiments, so far, are very promising on synthetic data. For example, we generate synthetic data according to the equation $y = \theta_1 * x + \theta_2$, where $\theta_1 = 1.0$ and $\theta_2 = 0.0$ (and many other values), run our implementation first without adding any noise (no differential privacy) and second with noise. In the first case, without noise, our implementation produces/predicts the value of $\theta_1 = 0.990$ and $\theta_2 = 0.0010$, very close to the actual. In the second case, with noise, we start with $\epsilon = 0.01$ and get $\theta_1 = 0.97924$ and $\theta_2 = 0.010800$ while with $\epsilon = 0.1$ we get $\theta_1 = 0.525030$ and $\theta_2 = 0.312111$. We stress that these are preliminary results and it requires more experimentation.

Algorithm 1 Constant Time Gradient Descent

1: **procedure** CONSTANT-DPGD($n, T, t, \theta_1^0, \theta_2^0, \gamma, \epsilon$) ▷ number of data points, number of iterations, clipping range, initial (guess) slope, initial (guess) intercept, learning rate, privacy parameter
2: Data: $\{(x_i, y_i)\}_{i=1}^n$
3: **for** `t := 1 to T` **do**
4: **for** `i := 1 to n` **do**
5: $\tilde{y}_i = \theta_1^t * x_i + \theta_2^t$
6: $\Delta_{i,t} = \begin{pmatrix} 2(\tilde{y}_i - y_i)x_i \\ 2(\tilde{y}_i - y_i) \end{pmatrix}$
7: $\Delta_{i,t}^c = \{\Delta_{i,t}\}_{-t}^{t}$
8: **end for**
9: $\Delta_t = \frac{1}{n}\sum_{i=1}^n \Delta_{i,t}^c + Lap(0, 4t/\epsilon)$
10: $[\theta_1^{t+1}, \theta_2^{t+1}] = [\theta_1^t, \theta_2^t] - \gamma * \Delta_t$
11: **end for**
12: return $[\theta_1^{T+1}, \theta_2^{T+1}]$
13: **end procedure**

```
int kl = l + h;
_(assert kl == 0 || kl == 1 || kl == -1)
int kr = abs(kl);
_(assert kr == abs_to_int(kl))
double retm = tau * kl + (1 - kr) * m;

_(apply bounded_lemma(l, h, kl, kr, m, tau, retm);)
// Proof that retm is within [-tau, tau]
_(assert -tau <= retm && retm <= tau)
gs->m = retm;
```

Figure 1. Gradient clipping snippet

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.

[2] Daniel Alabi, Audra McMillan, Jayshree Sarathy, Adam Smith, and Salil Vadhan. 2020. Differentially Private Simple Linear Regression. *arXiv preprint arXiv:2007.05157* (2020).

[3] Tiago Alves and Don Felton. 2004. TrustZone: Integrated Hardware and Software Security, White Paper. *ARM, July* (2004).

[4] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *Proceedings of the 11th*

USENIX Conference on Offensive Technologies (Vancouver, BC, Canada) *(WOOT'17)*. USENIX Association, USA, 11.

[5] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. 2017. Detecting privileged side-channel attacks in shielded execution with Déjá Vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 7–18.

[6] Raj Chetty, John Friedman, Nathaniel Hendren, Maggie R. Jones, and Sonya Porter. Working Paper. The Opportunity Atlas: Mapping the Childhood Roots of Social Mobility. Revise and Resubmit, American Economic ReviewPaper, Executive Summary, SlidesOnline web tool: https://opportunityatlas.org.

[7] Raj Chetty and John N Friedman. 2019. *A Practical Method to Reduce Privacy Loss when Disclosing Statistics Based on Small Samples*. Working Paper 25626. National Bureau of Economic Research. https://doi.org/10.3386/w25626

[8] Raj Chetty, Nathaniel Hendren, Patrick Kline, and Emmanuel Saez. 2014. *Where is the Land of Opportunity? The Geography of Intergenerational Mobility in the United States*. Working Paper 19843. National Bureau of Economic Research. https://doi.org/10.3386/w19843

[9] Gidon Ernst and Toby Murray. 2019. SecCSL: Security Concurrent Separation Logic. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 208–230.

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

[11] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961* (2018).

[12] Nick Hynes, Raymond Cheng, and Dawn Song. 2018. Efficient Deep Learning on Multi-Source Private Data. arXiv:1807.06689 [cs.LG]

[13] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. 2019. TensorSCONE: A Secure TensorFlow Framework using Intel SGX. arXiv:1902.04413 [cs.CR]

[14] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing. In *26th {USENIX} security symposium ({USENIX} security 17)*. 557–574.

[15] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy* (Tel-Aviv, Israel) *(HASP '13)*. Association for Computing Machinery, New York, NY, USA, Article 10, 1 pages. https://doi.org/10.1145/2487726.2488368

[16] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. *arXiv preprint arXiv:2104.14380* (2021).

[17] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. Cachezoom: How SGX amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 69–90.

[18] Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. 2020. CopyCat: Controlled Instruction-Level Attacks on Enclaves. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 469–486.

[19] Ivan Puddu, Moritz Schneider, Miro Haller, and Srdjan Capkun. 2020. Frontal Attack: Leaking Control-Flow in SGX via the CPU Frontend. In *30th USENIX Security Symposium (USENIX Security 21)*. 663–680.

[20] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. 2017. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs.. In *NDSS*.

[21] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2016. Preventing Page Faults from Telling Your Secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (Xi'an, China) *(ASIA CCS '16)*. Association for Computing Machinery, New York, NY, USA, 317–328. https://doi.org/10.1145/2897845.2897885

[22] Florian Tramèr and Dan Boneh. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. arXiv preprint arXiv:1806.03287. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1806.03287

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

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

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