IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 3, 24 pages.

https://doi.org/10.62056/akp-49qgxq

Slalom at the Carnival: Privacy-preserving Inference with Masks from Public Knowledge

Ida Bruhns[1], Sebastian Berndt[2], Jonas Sander[1] and Thomas Eisenbarth[1]

[1] Universität zu Lübeck, Lübeck, Germany
[2] Technical University of Applied Sciences Lübeck, Lübeck, Germany

Abstract. Machine learning applications gain more and more access to highly sensitive information while simultaneously requiring more and more computation resources. Hence, the need for outsourcing these computational expensive tasks while still ensuring security and confidentiality of the data is imminent.
In their seminal work, Tramèr and Boneh presented the Slalom protocol for privacy-preserving inference by splitting the computation into a data-independent preprocessing phase and a very efficient online phase. In this work, we present a new method to significantly speed up the preprocessing phase by introducing the Carnival protocol. Carnival leverages the pseudo-randomness of the Subset sum problem to also enable efficient outsourcing during the preprocessing phase. In addition to a security proof we also include an empirical study analyzing the landscape of the uniformity of the output of the Subset sum function for smaller parameters. Our findings show that Carnival is a great candidate for real-world implementations.

Keywords: Machine Learning · Secure Inference · Subset Sum

1 Introduction

Outsourcing of expensive computations is an important task in modern computer science. As calculations become more complex, maintaining the required hardware for every task is not feasible for smaller or medium-sized companies or individuals [NC23]. Furthermore, sharing insight and data between many parties can greatly enhance the usefulness of many applications. Different approaches on outsourcing computations are firmly established, "computing in the cloud" is used on a daily basis.

Computing on shared resources bears inherent security and privacy risks that are not a concern in other scenarios. Specifically, other tenants on the machine as well as the provider may tamper with data or computation or extract data. Simply requiring the user to trust the provider of the computation environment is not a reasonable solution, especially since many cloud service providers also earn money with the obtained data. Different techniques for this problem such as *fully homomorphic encryption* or *functional encryption* have been devised to minimize the trust needed and guarantee both the integrity of the computation and the confidentiality of data[CPTH21, HMSY21, NC23]. While these techniques provide a sound theoretical solution, the resource costs to actually deploy them in practice are prohibitively high. To keep resource consumption and computational overheads reasonable, more specialized solutions have been developed for important tasks.

Acknowledgments: The research received funding from the BMBF and EU in the project AnoMed. We would like to thank Florian Tramèr for his friendly advice regarding the Slalom source code.

E-mail: ida.bruhns@uni-luebeck.de (Ida Bruhns), sebastian.berndt@th-luebeck.de (Sebastian Berndt), j.sander@uni-luebeck.de (Jonas Sander), thomas.eisenbarth@uni-luebeck.de (Thomas Eisenbarth)

Received: 2024-07-09 Accepted: 2024-09-02

One of the most resource intensive applications nowadays revolves around the field of *machine learning*. It has been in the scientific and economic spotlight for over a decade now and deep neural networks are known to require a large amount of computing power. The two most common tasks here are the *training* of a neural network and the *inference* or evaluation of such a trained network. As training requires extensive amounts of data and computational resources, machine learning as a service (MLaaS) is a booming field: Smaller parties can train models on external hardware, since training resources are only required for a relatively short amount of time, they can utilize other parties data in shared learning, or simply utilize a readily trained model to run inference on their data [DFH+24]. However, outsourcing computation may raise ethical or legal issues depending on the processed data. It is easy to see why data sets such as health data, credit statements, or a companies intellectual property are not suitable to be processed on a shared, insecure machine without further protection. Both privacy preserving training and inference are active research fields, working to protect the training and inference data as well as the model and the output [LFFJ20, NC23, DFH+24, ZLT+24, ELD24].

In this work, we focus on the problem of hardware-assisted secure outsourced inference for neural networks (NN). A model owner has spent resources to train and refine a model. The neural network $\mathsf{NN}\colon \mathcal{X} \to \mathcal{Y}$ consists of layers $\mathsf{NN}_i\colon \mathcal{X}_i \to \mathcal{Y}_i$ and each layer consists of a linear transformation Lin_i followed by a non-linear activation function NLin_i. The linear layers are usually matrix multiplications or convolutions. As convolutions can be converted to a matrix multiplication, we will use the matrix multiplication as the exemplary function in this work. Furthermore, we assume that the input $\mathcal{X}_i$ and the output $\mathcal{Y}_i$ are represented by Integer vectors. The model owner uses a server S to host NN and now offers inference as a service to clients. This has two reasons:

- The model owner does not want to share NN, which is a business asset [1].
- The client C on the other hand does not have the computational resources to infer on the model, and might even lack the knowledge to deploy it.

The client C provides the input data $\mathsf{x} \in \mathcal{X}$, which is sensitive information not to be shared with S or other tenants. The goal of the client is to compute $\mathsf{NN}(\mathsf{x}) \in \mathcal{Y}$ with minimal computation costs by making use of S without revealing information about the input x or the output $\mathsf{NN}(\mathsf{x})$ to the server or model owner. This is called *input privacy* and *output privacy* respectively.

To achieve privacy for the client, the model owner offers a trusted execution environment (TEE) within the server. The client can load the data into the TEE, which is then protected from a potential malicious model owner. As usual, the code within the TEE can be checked by the client via remote attestation and the data processed within the TEE is constantly encrypted in memory. Inferring purely on the TEE would however create massive performance issues, either increasing the MLaaS prices or rendering the model owners business model useless. One approach to handle this problem is by adding a *fast processing unit (FPU)* to the server. This can be a GPU or specialized hardware such as FPGAs.

The scenario is depicted in Figure 1.

This creates the challenge for the model owner to move as much computational load as possible to the FPU *without* compromising the client data. In order to protect the client's data, some operations still need to be computed in the TEE, which is slower than the FPU by orders of magnitude. Hence, the main question of this line of research is to minimize the amount of computation required in the TEE while still guaranteeing privacy to the client.

[1] We note here that strong model extraction attacks exist (e.g., [CJM20, PMG+17, TZJ+16]) and thus do not focus on model privacy in this work.

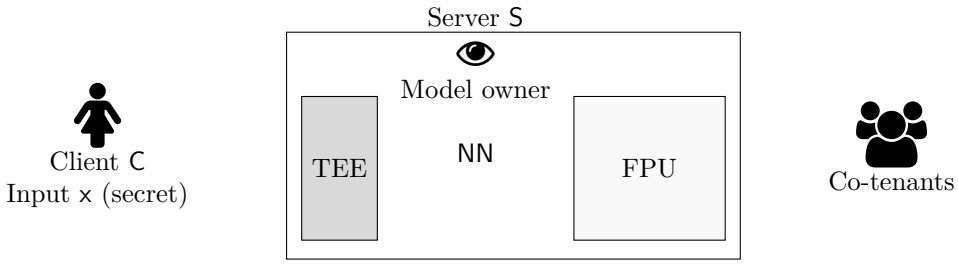

Figure 1: Components and parties in the secure inference scenario. The server S contains both a trusted execution environment (TEE) and a fast processing unit (FPU) such as a GPU. The client C provides the secret input x. The network NN is known to the model owner.

1.1 Slalom

One solution to outsource the computational load to the FPU was presented by Tramèr and Boneh [TB19], who developed a framework called Slalom that allows a private computation of $\mathsf{NN}(\mathsf{x})$ if the server is equipped with a secure TEE. To do so, they use the *offline-online* model inspired by very efficient MPC protocols such as BDOZ [BDOZ11] or SPDZ [DPSZ12]. In this model, the computation is split into an offline or *preprocessing* phase that is independent of the sensitive input x, where the TEE generates some masking values and their counterparts (later referred to as unmasking values) for the client. These values, combined with x are then used in the online phase to produce the output $\mathsf{NN}(\mathsf{x})$.

The approach taken by Slalom works well if the offline phase is not included in the total runtime of the framework, and indeed Slalom is considered one of the milestones in private inference and many future approaches rely on trusted hardware to build fast and secure solutions [MSK+20, HWA22, NAA22, SBBE23, WZB+23]. It does however contain several problems:

Total Runtime: In the Slalom setting, *the entire network is evaluated on the masks* in the TEE during the preprocessing phase. As the masks have the same size as the input, the preprocessing phase plus the online phase take *more* time than just inferring the entire network on the TEE to begin with.

Implementation issues: Performing the preprocessing phase on TEEs may lead to memory issues as enclave size needs to be appointed before the enclave is created and swapping out the memory massively lowers the performance. In addition, it must already be known in the offline phase how many inputs need to be processed in the online phase for the masks precomputations.

Idle FPU: During the preprocessing phase, only the slow TEE works, while the much more powerful FPU is idle. As the server owner bears the cost of generating the unmasking values in the TEE during the preprocessing phase, she has an inherent interest in an efficient use of resources.

Running out of Randomness: Slalom is only suitable for quite restricted situations with sufficient time between the online phases that can be used to prepare sufficient preprocessing material. If the time between two online phases is too short, Slalom will run out of randomness, leading to a large performance penalty.

It is unclear if the authors were aware of these complications, as they did not implement the offline phase in the TEE in their PoC, and rather chose to do it insecurely on the

FPU[2].

In this work, we design a method to perform the preprocessing phase more efficiently using the Subset sum problem over finite fields as a randomness generator.

1.2 The Subset Sum Problem

The Subset sum problem is a well-known NP-hard problem. Given a finite field F_p and a set $S = \{s_1, \ldots, s_n\} \subseteq F_p^n$ of field elements and a function

$$f(k) = \sum_i k_i S_i \mod p$$

that maps a binary vector k to another field element, the goal is to reconstruct the binary characteristic vector k. From a cryptographic perspective, $f(k)$ is strongly believed to be a *one-way function* (for appropriate choice of the parameters n and p), meaning that there is no efficient algorithm to recover k from $f(k)$. Furthermore, f can be used as pseudo-randomness generator under certain conditions [IN96, Bri84, MM11], which Carnival leverages to generate pseudo-random one time pads (OTP) as masks. We will give a detailed description of the process in Section 4. The high-level improvement is the swapping of matrix multiplication in the TEE for matrix additions only, which reduces the computational complexity and runtime drastically. As an added bonus, the FPU is utilized during the setup phase as well, leading to better resource balancing.

1.3 Contribution

The main result of our paper is a secure method to perform pre-calculations on untrusted hardware. We use the Slalom algorithm as an example. In this work, we

- Propose Carnival, a way to use publicly performed calculations to generate secret masks for confidential input values.
- We prove that our system is secure under the assumption that the Subset sum problem over finite fields is a one-way function, a longstanding cryptographical assumption.
- As a case study, we propose Slalom at the Carnival (S@C), a new variant of Slalom with an improved preprocessing phase and an analysis of the performance gain. Even when choosing conservative parameters, this already gives a speedup between 2.2 and 11 in the provable scenario.

Our approach makes better use of the FPU by putting it to work during the setup phase as well as the online phase. While we demonstrate it with the Slalom use case, the Carnival primitive can be applied in many settings.

2 Background

This section provides the reader with background information on knapsack functions, trusted execution environments, quantization, outsourced computations in ML settings, and security definitions.

[2]This is apparent from the code and reflected in the comment "This is obviously insecure, but we currently compute the unblinding factors outside of the enclave for simplicity" [Tra19].

2.1 Knapsack Functions in Cryptography

The Subset sum problem is a specific variant of the knapsack problem, namely a 0-1 knapsack problem where the weights of the knapsack items equal their profit [Pis05]. Knapsack functions have a long history as cryptographic functions. Many cryptographic systems based on this function family have been introduced [Lem79, MH78, KG11, CR88]. While knapsack functions lead to one-way functions with very useful properties [Mic07], using them in asymmetric scenarios has been a very difficult task, as shown by many broken systems [Odl90, Sha82, OT04]. One of the most useful such properties is the fact that knapsack functions can be used as pseudo-random number generators or hash functions [IN96], with a number of properties that make them interesting candidates for various practical scenarios:

- Knapsack functions are able to generate pseudo-randomness relatively easy, using additions only.
- The highly parallel instructions over a finite field can be implemented very efficiently and
- Linear operations on the knapsack elements result in linear transformations of the resulting one-time pad, making them suitable for homomorphic operations.

Most of the asymmetric knapsack cryptosystems devised so far use a specially constructed set S as the public key and additional information about S as private key. In these systems k is the plaintext and $f(k)$ is the ciphertext [Lem79, MH78]. As usual in public key crypto systems, the ciphertext can only be deciphered efficiently with the private key, namely the additional information about S. Hence, the trapdoor information is concealed in S [LO85].

In contrast, we do not use S to conceal any information, but use k as the key to generate an OTP which is then used to encrypt the message, generating a symmetric cryptosystem or masking scheme.

2.2 Trusted Execution Environments

TEEs provide a secure environment on untrusted hardware by offering hardware-sealed, attested enclaves. In these, everything from a small subroutine to a whole VM can run securely, undisturbed even by a compromised operating system or hypervisor. Common TEEs are Intel Software Guard Extensions (SGX) [MAB+13, CD16, Int21], AMD Secure Encrypted Virtualization (SEV) [Kap16], and Sanctum [CLD16]. The ARM Confidential Compute Architecture (CCA) [LLD+22] and Intel Trust Domain Extensions (TDX) [SMF21] are still in development. TEE offers memory isolation and attestation for both the data and the code deployed within a specific enclave. This allows multiple mutually distrustful parties to compute on untrusted hardware while using sensitive data [CB19]. While there has been some discussion about the future relevance of TEEs after Intel discontinued SGX on consumer devices, the efforts of both Intel and other vendors to build alternatives is evidence that TEEs will be an important feature in the future.

2.3 Quantization

ML models usually have floating point weights, biases, intermediate values and sometimes input values, and GPUs also work on these values. However, to guarantee perfect security, masking only works on values of a finite field. Hence, floating point values need to be transformed into such integer values, then masked and re-transformed into floating point values constantly during both Slalom 's and Carnival's runtime, a process called

quantization. We used the same quantization as Slalom , which is derived from Gupta et al. [GAGN15]: Floating point numbers x are converted to integers as $\hat{x} = \mathsf{FP}(x; l) := \mathsf{round}(2^l \cdot x)$. A linear layer L with kernel W and bias b is thus quantized with the parameters $\hat{W} = \mathsf{FP}(W, l)$ and $\hat{b} = \mathsf{FP}(\mathsf{b}, 2\mathsf{l})$. After inferring L on input x, the output needs to be scaled by 2^{-l} and rounded to an integer again. For more details, we refer to [TB19].

2.4 Secure Outsourced Computation

Outsourcing computations to shared machines is a natural and sensible way to use resources: not every party can (and should) have the resources for any necessary computation on premise, maintaining a lot of different specialized hardware is not practical and efficient, and using optimized implementations on shared machines is beneficial. In the field of machine learning, models are usually trained by one party and then used by others, and classification tasks may not come in at a steady flow but in bursts. This means that it makes sense to outsource the classification tasks to a MLaaS server holding the model. This scenario (also depicted in Figure 1) offers no inherent protection of the input values provided by the clients. There are three main possibilities to add this protection, all of which come with a performance penalty and sometimes also affect accuracy:

Homomorphic Encryption (HE) As linear network layers such as convolutional or fully connected layers can be seen as a series of multiplications by a constant and additions, which are homomorphic operations, they can be calculated with relatively little overhead [GDL+16, CBL+18]. The non-linear activation layers like softmax or ReLu layers however are not easy to calculate using homomorphic encryption. They are usually approximated with low-degree polynomials [CKKS17] or adapted cosine functions [WMW23] in fully homomorphic approaches. Both approximations come at an accuracy loss and sometimes require extensive retraining of the network. The scheme of choice for machine learning is currently the Cheon-Kim-Kim-Song (CKKS) scheme, which works on polynomial quotient rings and supports batch evaluation [CKKS17, NLDD23]. The inaccuracy caused by the approximated activation functions grows with the network depth: The largest neural network evaluated using purely HE on the CIFAR-10 dataset is a net with 10 convolutional layers [NLDD23].

Secure Multiparty Computation (SMPC) Usually, a form of oblivious transfer (OT), garbled circuits (GCs) or secret sharing is implemented in SMPC based schemes. Mohassel and Zhang [MLS+20] developed SecureML in 2017, where two untrusted servers train a model using 2-party MPC techniques. Just as the HE approaches, they approximate the activation functions sigmoid and softmax to reduce complexity, but they also use GCs to further speed up these functions. ABY^3 [MR18] is a variant of SecureML for three servers. The scheme Minionn combines secret sharing with GCs in a one server setting, again with approximated activation functions. Both these schemes and Deepsecure [RRK18], which uses binary circuits for garbling and oblivious transfer for non-linear layers, work on networks with few layers, but suffer from the huge overhead each activation layer adds. In the end, they can all only work with networks with 3 layers or less. Gazelle improved on this boundary by using HE in the linear layers and GCs in the activation layers [JVC18] while optimizing packaging and encryption between the layers. The techniques from this scheme where adapted to the GPU by the Delphi scheme [MLS+20]. The HE computations are moved to an offline phase again, which is also included in the performance analysis. The offline phase uses up the vast majority of the runtime. Recent approaches such as Dash in 2023 or Piranha in 2022, add usability as a distinguishing feature as well as performance [SBBE23, WWP22]. Dash allows both model owners and clients a low-

threshold entry to efficient garbled circuits by enabling model loading from standard files, without deep knowledge of GCs. The Piranha platform follows a similar approach and enables developers of secret-sharing-based MPC schemes to leverage a GPU without knowledge of GPU programming. In both cases models do not need to be retrained.

Trusted Hardware As mentioned above, TEEs are a viable tool to protect privacy on shared hardware. The main drawbacks are the need for specialized hardware and resource limitations e.g. through paging mechanisms. The code within the enclave is obviously victim to any vulnerabilities in the enclave code or concept itself. Another interesting problem is the verification of the attested code: Depending on the problem at hand, the client may not be able to fully verify the code within the enclave. This can be circumvented by passing out intermediate results and verifying them on a random sample basis. Conceptually, TEEs provide code confidentiality and integrity as well as remote attestation capabilities through logic isolation and cryptographic protection [CD16, Int24]. Over the years, the research community has demonstrated a variety of different side-channel attacks on current implementations of TEEs, especially on SGX [XCP15, BMD+17, BPS17, MIE17, BMW+18, MOG+20]. Note that side channel attacks are implementation attacks. Just as cryptographic implementations, TEE-based implementations may need hardening against these attacks. If the implementation is not hardened, it will be insecure, whether it is a cryptographic implementation or a TEE-protected implementation [WRPE24]. More recent proposals such as Keystone [LKS+20] and Intel TDX [Int24] also take hardware-based side-channels into account, while other potential weaknesses need to be addressed by hardening implementation, e.g. through constant-time implementation approaches [BBG+20]. Therefore, SGX, combined with traditional software-based efforts, can significantly raise the bar for successful attacks.

Some approaches use a combination of the above, trying to overcome the challenges of one approach by using another. For example, CrypTFlow is a TEE-MPC hyrbrid solution [KRC+20]. Indeed, the masking operation in S@C can be considered a form of homomorphic encryption, while the mask generation is performed in the TEE.

Formal Definitions We will consider outsourcing the computation of $g(x)$ for a sensitive value x and a publicly known function g to a non-trusted party. We follow and extend the definitions by Tramèr and Boneh [TB19]. A *secure outsourcing scheme* for a function $g\colon \mathcal{X} \to \mathcal{Y}$ between a client C and a server S consists of three algorithms Setup, $\mathsf{Preproc}$, and Online. The algorithm Setup is called a single time and produces some public parameters param. Given these parameters, $\mathsf{Preproc}$ can be used by C to produce some data-independent state state. Finally, Online is an interactive protocol between C providing the inputs param, state, and the sensitive information $\mathsf{x} \in \mathcal{X}$ such that at the end of the protocol, C receives a value $y \in \mathcal{Y}$ or aborts the protocol. We denote the result of Online obtained by C when using inputs param, state, and x and running S by $\mathsf{Online}_{\mathsf{C},\mathsf{S}}(\mathsf{param}, \mathsf{state}, \mathsf{x})$. Such a scheme needs to fulfill several important properties such as:

Correctness: For any param produced by Setup, any state produced by $\mathsf{Preproc}(\mathsf{param})$, any $\mathsf{x} \in \mathcal{X}$, and any $y = \mathsf{Online}_{\mathsf{C},\mathsf{S}}(\mathsf{param}, \mathsf{state}, \mathsf{x})$, we have $y = g(\mathsf{x})$[3].

Privacy: For any param produced by Setup, any state produced by $\mathsf{Preproc}(\mathsf{param})$, any $\mathsf{x} \in \mathcal{X}$, any $\mathsf{x}' \in \mathcal{X}$, and any probabilistic polynomial-time algorithm S^*, the views of S^* in $\mathsf{Online}_{\mathsf{C},\mathsf{S}^*}(\mathsf{param}, \mathsf{state}, \mathsf{x})$ and in $\mathsf{Online}_{\mathsf{C},\mathsf{S}^*}(\mathsf{param}, \mathsf{state}, \mathsf{x}')$ are computationally indistinguishable.

[3]Similar to Slalom, to prevent a malicious server from using a different function $g' \neq g$, we could make use of zero-knowledge proofs.

t-**Integrity:** For any param produced by Setup, any state produced by $\mathsf{Preproc}(\mathsf{param})$, any $\mathsf{x} \in \mathcal{X}$, and any probabilistic polynomial-time algorithm S^*, the probability that $\mathsf{Online}_{\mathsf{C},\mathsf{S}^*}(\mathsf{param}, \mathsf{state}, \mathsf{x}')$ does lead to a value $y' \in \mathcal{Y}$ with $y' \neq g(\mathsf{x})$ is at most t.

3 The Baseline: Slalom in detail

Slalom is a hybrid scheme for outsourced machine learning inference that elegantly combines homomorphic encryption using one-time pads for the linear layers and a TEE for the non-linear layers. It guarantees the integrity of the inference and has an optional privacy feature for input and output privacy. We only consider the privacy scenario. Slalom works in the setting described in Figure 1. The general approach of Slalom is illustrated in Figure 2 and can be summed up as follows:

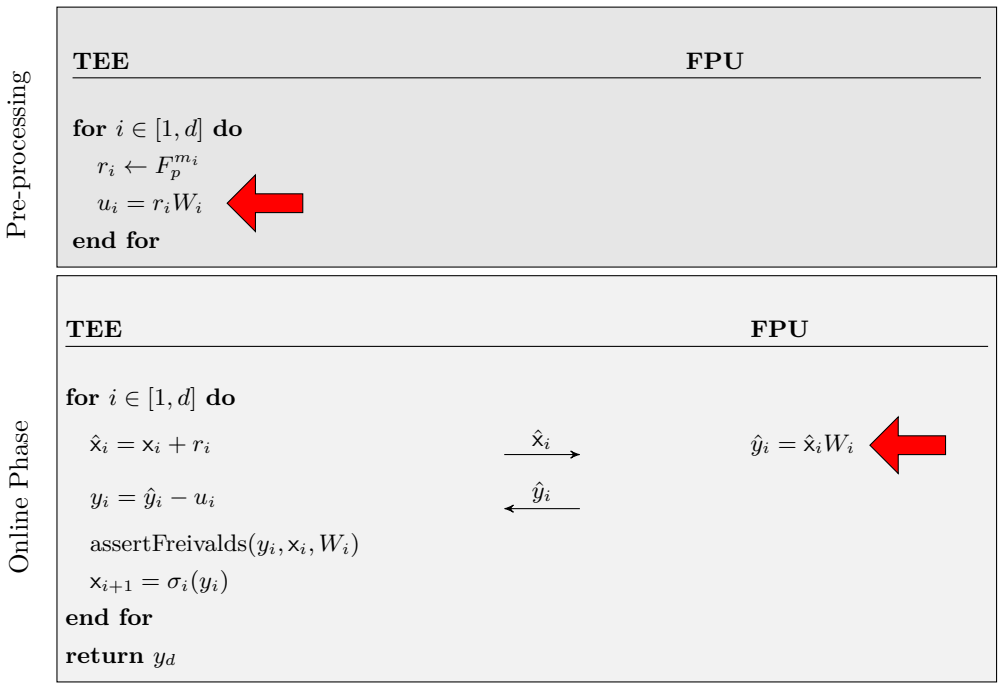

Figure 2: The Slalom algorithm by Tramèr and Boneh [TB19]. Note that the marked calculations are essentially the same. Additionally, the FPU is idle during the preprocessing phase.

The TEE is used to perform the entire preprocessing phase, generating the random masks and calculating the corresponding unmasking values. This means that it infers the entire network repeatedly on random values. The FPU is idle during the preprocessing phase. In the online phase, the client C sends their secret input x to the server S, where it is received and masked by the TEE. The masked values are then passed to the FPU, which calculates the linear layer. The masked result is passed back to the TEE, where the corresponding unmasking value is used to obtain the result of the linear layer. The following non-linear layer is then applied to this result, generating the input for the next linear layer. This is then masked, send to the FPU and so on.

In a more formal way, the two phases work as follows:

preprocessing Phase: For each layer NN_i of NN, the TEE generates a *masking value* $r_i \in \mathcal{X}_i$ uniformly drawn from $\mathcal{X}_i$ and computes the *unmasking value* $u_i = \mathsf{Lin}_i \cdot r_i$.

The preprocessing phase is performed by the TEE, as it generates the masking values, which need to be hidden from the server owner and model owner.

Online Phase: To evaluate layer NN_i on secret input x_i, the TEE computes $\hat{\mathsf{x}}_i = \mathsf{x}_i + r_i$ and sends $\hat{\mathsf{x}}_i$ to the server. The FPU then computes $\hat{y}_i = \mathsf{Lin}_i(\hat{\mathsf{x}}_i)$ and sends $\hat{y}_i$ to the TEE. The TEE now computes $y_i = \hat{y}_i - u_i$ and sends $\mathsf{x}_{i+1} = \mathsf{NLin}_i(y_i)$.

Using this approach, one can show the input privacy of Slalom by making use of the observation that the masking provides information-theoretic security.

Theorem 1 (Theorem 3.2 in [TB19])**.** *Assume that all random values are generated using a secure PRNG with security parameter λ. Then, Slalom is a secure outsourcing scheme guaranteeing correctness and privacy.*

As mentioned in Section 1, the preprocessing phase generates several problems such as performance-intensive calculations within the TEE while the FPU is idle, possible implementation issues within the TEE for large networks and the possibility to run out of masking and unmasking values if the offline times are not long enough. We would like to stress that none of these issues affects the correctness or security of Slalom.

4 The Carnival Primitive: Building OTPs from Public Randomness

In this section we formally define the cryptographic primitive behind Carnival. In general, the goal is to let an untrusted entity evaluate the function $g(x) := g_\alpha(x)$ on a private input x and public (for both parties) parameters α. While we will discuss this task in a more general setting, it is instructive to first consider $g(x)$ as one layer of a neural network, i.e., $g(x) = \mathsf{NLin}(\mathsf{Lin}(x))$.

We will present two different approaches to select appropriate parameters. In the first approach, we show how to obtain a provable secure version of our primitive, based on the hardness of the Subset sum problem and the state-of-the-art attacks against it.

4.1 Masking Inputs

In order to keep x private, we first compute the value $h(x, k)$ that *masks* the value x via some secret key k. The untrusted party is then given $h(x, k)$, computes $g(h(x, k))$, and then returns this result to us. Finally, to obtain the desired result $g(x)$, we now need to retrieve it from $g(h(x, k))$.

We only consider masking functions $h(x, k)$ that use an *additive* masking, i. e. $h(x, k) = x + f(k)$ for some function f. Now, if $g(x)$ is linear, we have $g(h(x, k)) = g(x + f(k)) = g(x) + g(f(k))$. Hence, we only need to compute $g(f(k))$ to obtain $g(x)$. In the easier model of Slalom, where the costs for preprocessing are ignored, we could compute $g(f(k))$ by ourselves in the preprocessing phase, as $f(k) = k$ here and the neural network described by g is public. But, by choosing $f(k)$ more carefully, we are able to reduce these preprocessing costs drastically.

Using Subset sum: Now, suppose that $f(k)$ is defined via a Subset sum problem. Recall that in Section 1, we defined our Subset sum problem over a finite field F_p as a set $S = \{s_1, \ldots, s_n\} \in F_p^n$ of n field elements and a function

$$f_{S,p}(k) := f_p(k, S) = \sum_i k_i s_i \bmod p$$

that maps a binary vector k to another field element.

The parameters p and S (and therefore by definition also $n = |S|$) are public, while k is the private key. In order to facilitate the pseudo-randomness of the Subset sum problem, we will first sample the set $S = \{s_1, \ldots, s_n\}$ randomly from F_p. Note that this process can also be performed in public (as long as the random choice of S is guaranteed). Now, we can compute $u_i = g(s_i)$ for $i = 1, \ldots, n$ in public (we later discuss how to verify this computation).

In the preprocessing phase, the client can choose a new random key k and compute $r = \sum_i k_i s_i \bmod p$ and $u = \sum_i k_i u_i \bmod p$. Note that the preprocessing phase thus only requires the addition of $2n$ field elements, but not computation of g.

In the online phase, the client only needs to calculate $\mathsf{x} + r$ and is given $g(\mathsf{x}) + g(f(k))$, from which it subtracts u. Applying linearity of g again, we see that for $f = f_{S,p}$, we have

$$g(f(k)) = g(f_{S,p}(k)) = g\left(\sum_i k_i S_i \bmod p\right) = \sum_i k_i g(s_i) \bmod p = u,$$

as k_i is only a binary scalar. Hence, during the online phase, the client is able to compute $g(\mathsf{x})$ using only two additions.

One of the main advantages of our approach is the fact that we can use the same knapsack instance, generated during the setup phase, for a large number of preprocessing and online phases. We only need to guarantee that the preprocessing phase produces a fresh key. Hence, a single setup with n items can be securely used for roughly $2^{n/2}$ preprocessing and online phases (at which point a key will probably repeat).

4.2 Provable Security

In this subsection, we will show that our approach is provable secure as long as the underlying Subset sum problem is sufficiently hard. Clearly, there are two parameters that influence the security of the Subset sum problem and thus of our approach. The first parameter is the size of the individual field elements. In our scenario, this size is fixed by the size of the elements of the neural network. Hence, we will focus our attention on the second parameter, the number n of different items S_i in the Subset sum instance. For performance reasons, it is desirable to keep n as small as possible without compromising the security of the system.

We will denote the security parameter by λ and abbreviate the term probabilistic polynomial time by PPT. A sequence $\{p_\lambda\}_\lambda$ is *negligible* if, for all $c \in \mathbb{R}_{>0}$, there is $\lambda_0 = \lambda_0(c)$ such that $p_\lambda < 1/\lambda^c$ for all $\lambda \geq \lambda_0$, i.e., it is smaller than the inverse of every polynomial. We call a function ensemble $\{f_\lambda\}_\lambda$ with $f_\lambda \colon D_\lambda \to R_\lambda$ to be *one-way*, if $f_\lambda(x)$ is computable in polynomial time and

$$\Pr_{x \leftarrow D_\lambda}[f_\lambda(\mathcal{A}(f_\lambda(x))) = f_\lambda(x)]$$

is negligible for all PPT attackers $\mathcal{A}$. Hence, an attacker that is given the value $f_\lambda(x)$ should not be able to construct a value x' (not necessarily identical to x) such that $f_\lambda(x') = f_\lambda(x)$. Finally, we call such a function ensemble $\{f_\lambda\}_\lambda$ a *pseudo-random generator*, if $f_\lambda(x)$ is computable in polynomial time, $|R_\lambda| > |D_\lambda|$, and if

$$|Pr_{x \leftarrow D_\lambda}[\mathcal{A}(f_\lambda(x)) = 1] - Pr_{y \leftarrow R_\lambda}[\mathcal{A}(y) = 1]|$$

is negligible for all PPT attackers $\mathcal{A}$: An attacker that is either given a completely random element from R_λ or $f_\lambda(x)$ for a completely random element from the smaller set D_λ should not be able to distinguish between these scenarios.

From a purely asymptotic point of view, there is a very close relation between the one-wayness of the knapsack function and its pseudo-randomness, already established by Impagliazzo and Naor [IN96].

Theorem 2 (Theorem 2.2 in [IN96])**.** *If the Subset sum function ensemble* $\{f_p\}_p$ *with* $f_p\colon F_p^n \times \{0,1\}^n \to F_p^n$ *with* $\log(p) \geq (1+\epsilon)n$ *for some* $\epsilon > 0$ *is one-way, then it is also a pseudo-random generator*[4]*.*

This is already sufficient to prove the security of Carnival, following the security definitions of Tramèr and Boneh.

Theorem 3. *Let Carnival be the above protocol to compute a linear function* g*. Assume that all random values are generated using a secure PRNG with security parameter* λ *and that the knapsack function* $f = f_{p,S}$ *is a one-way function chosen with security parameter* λ*. Then, Carnival is a secure outsourcing scheme guaranteeing correctness and privacy.*

Proof. The correctness follows easily from our discussion above. Theorem 2 implies that the one-wayness of f also guarantees pseudo-randomness. Then, the pseudo-randomness of f means that we can replace $f(k)$ by a completely random value r. This implies privacy, as the runs of $\mathsf{Online}_{\mathsf{C},\mathsf{S}^*}(\mathsf{param}, \mathsf{state}, \mathsf{x}+r)$ and $\mathsf{Online}_{\mathsf{C},\mathsf{S}^*}(\mathsf{param}, \mathsf{state}, \mathsf{x}'+r)$ are identical. □

Example: Outsourced Matrix Multiplication We illustrate the primitive with the following scenario: A client wants to multiply a private vector x with a matrix W. Since this is computationally expensive, the client would like to outsource the matrix multiplication to a server. The input vector should of course remain private. In this scenario, $g(x) = x \cdot W$. Since $g(x+m) = g(x) + g(m)$ it is possible to let the server generate many random masking vectors m_j and calculate the corresponding $g(m_j)$. The masks m_i are then used as S to generate the OTP r and the unmasking value u for the OTP is calculated accordingly.

Figure 3 shows the entire preprocessing phase including outsourced generation of a set S_m of masking and a corresponding set S_u of unmasking values by the server and the required processing step by the client to obtain the secret OTP for masking and unmasking. Note that the client only has to perform additions. After the preprocessing, the client would add an OTP to the input value x and send the masked input to the server. Once the client receives the masked result, it subtracts the corresponding unmasking value to obtain the result.

Choice of Parameters While the above result already implies the security of our construction, it only gives an asymptotic bound that might not be meaningful for smaller, practically relevant parameters. Furthermore, the reduction from one-wayness to pseudo-randomness relies on the Goldreich-Levin theorem [GL89], which has a large running time and is thus probably not tight.

In the following, we thus discuss the choice of practically relevant security parameters that take the best known attacks into consideration. To the best of our knowledge, the current state-of-the-art attack against the Subset sum problem is due to Bonnetain et al. [BBSS20] and runs in time $\tilde{O}(2^{0.283n})$ for field size $p \approx 2^n$. Hence, in order to obtain 128-bit security by using 2, we need $n \geq 453$. Using this parameter, we obtain a cryptographically secure protocol.

We stress here that using smaller parameters of n only show that the attack of Bonnetain et al. [BBSS20] is able to break the one-wayness of the Subset sum problem. This however does not imply that the distribution generated by the Subset sum instance is far from uniform. Consider, e.g., the field elements $S = \{s_0, \ldots, s_{n-1}\}$ with $s_i = 2^i$ and $p \approx 2^n$. For a random key choice k, the value $f_{S,p}(k)$ is clearly uniformly distributed, although the underlying Subset sum problem is trivial (as the solution can be directly deduced from the single bits of $f_{S,p}(k)$).

[4] As [IN96] does not consider one-way functions with public parameters, the function should also output S, but we ignore this here for the sake of readability.

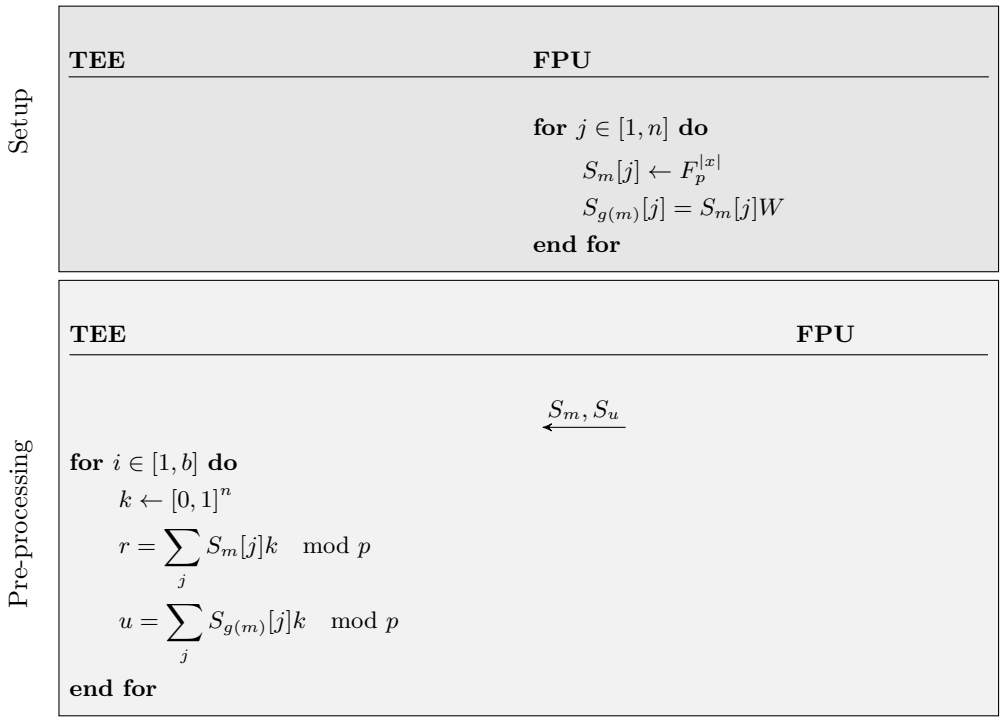

Figure 3: Illustration of the setup phase and one corresponding preprocessing phase in Carnival: Note that the FPU is generating most of the randomness and the communication is unilateral. Furthermore, a single setup phase can correspond to many preprocessing phases. One preprocessing phase is required for each of the b expected batches, but they can all be computed ahead of the online phase.

5 Slalom at the Carnival: Integration in Slalom Framework

We consider an inference-as-a-service scenario where clients send inputs and get the corresponding results. The inputs x and the results $\mathsf{NN}(\mathsf{x})$ are supposed to stay private while the machine learning model NN consisting of the network architecture, the weights W and the activation functions σ are known to the FPU. In this scenario, the client has to cover the cost of the precomputation phase. Thus, an expensive offline phase will directly result in cost for the client. We focus on the Slalom framework devised by Tramèr and Boneh for private inference [TB19]. While the Slalom approach is very efficient in the online-phase, the preprocessing phase generating the unmasking values is very expensive.

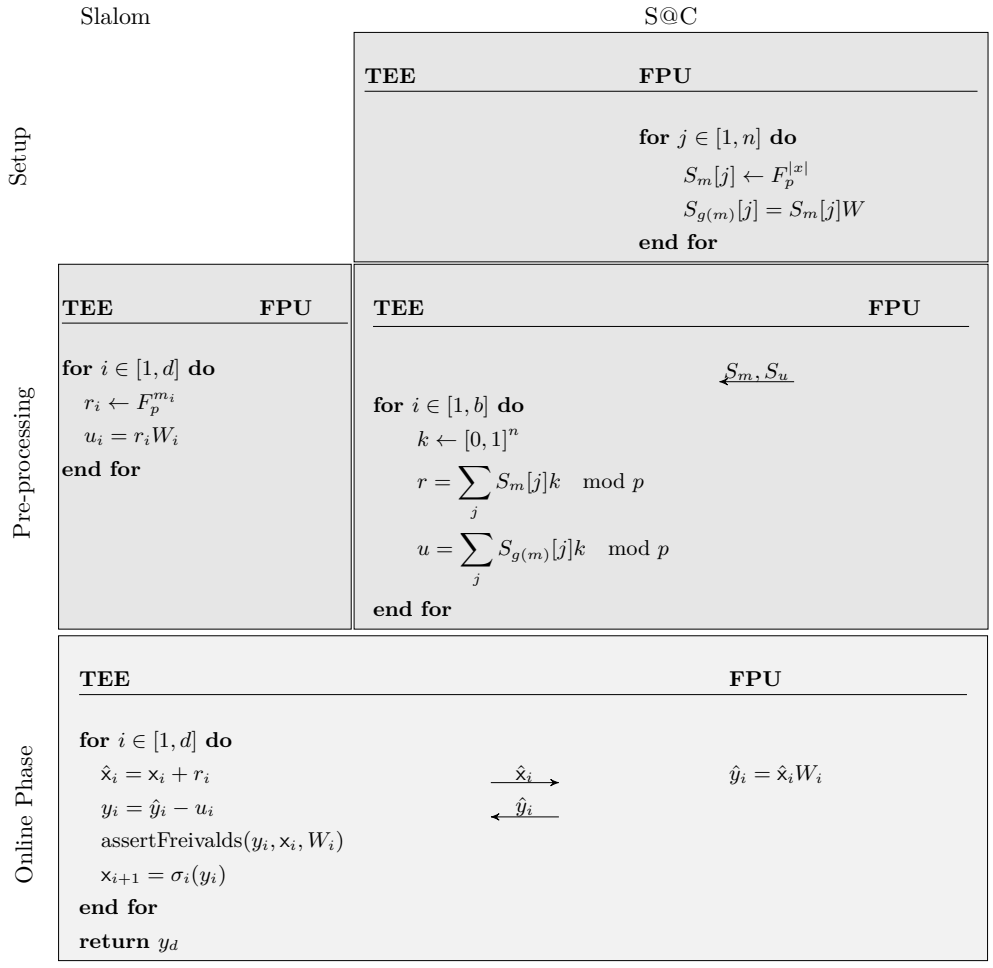

Figure 4: Slalom pseudocode for both variants for a network with d layers. While the preprocessing phase of our variant is longer, it significantly reduces the computational load of the TEE by outsourcing more work to the FPU. The online phase is the same in both protocols except the assertFreivalds step is optional in S@C if integrity is not a security goal.

As shown in Figure 4, in the original Slalom the TEE masks the input values with random numbers, sends the blinded values to the FPU and unblinds the result of every layer with pre-calculated unmasking values: for each weight matrix W_i, a random vector r_i is sampled and $u_i = r_i W_i$ is calculated in the TEE. The overall effort for the offline-

and online-phase is thus higher than just inferring the entire model in the TEE. While splitting the computational performance analysis into an offline- and an online phase seems to be common practice in machine learning settings, a costly offline phase is still a relevant factor in many settings. Slalom's threat model assumes that NN is known to the FPU and private inputs and outputs. So while all W and b as well as the activation functions are public, x and y stay secret.

A second problem with the Slalom approach is that the number of unblinding factors computed in the offline phase is limited, which directly implies that the system can run out of unblinding factors in the online phase. This would require shutting down the system to launch a new offline phase, which again is costly in both time and money.

Carnival can be used to overcome both of these challenges: It uses the FPU to generate many potential masks and the corresponding unmasking values. With appropriate scaling and buffering, the setup phase on the FPU can be parallelized for several layers, increasing the performance further. With the public set S of possible masks, the TEE can then pick a random key k to add up some of the masks and generate a secret one. The corresponding unmasking values need to be summed up as well. As shown in the previous sections, a polynomial time attacker has no efficient way of telling which of the provided masks were used to build the final mask, and thus cannot determine the private user input due to the pseudo-randomness of the knapsack function. The particular property of the Carnival scheme leveraged here is the fact that linear transformations on the elements of set S lead to linear transformed OTPs, allowing a lot of formerly secret computation to be moved to the FPU.

5.1 Performance Analysis and Comparison

The complexity of a conventional 2D convolution is quadratic with three hyperparameters: number of channels C, kernel size K, and spatial dimensions of the input H and W, and its computational complexity is $\mathcal{O}(C^2K^2HW)$. An addition of a matrix of the same dimensions can be performed in $\mathcal{O}(CKHW)$. In S@C, the TEE needs to generate one random n bit binary number and then perform a maximum of n additions followed by modular reductions. The expensive operations of convolutions and the generation of large amounts of random numbers are shifted to the FPU, and the workload of the much slower TEE is reduced by orders of magnitude. Assuming that the FPU is f times faster than the TEE, a performance gain is achieved if $n < f$. Since n is 453 in the cryptographically secure scenario, S@C is faster than Slalom if $f > 453$ or, for the statistically secure scenario, if $f > 2l + 1$.

As mentioned in the introduction, there is no implementation of Slalom available for comparison. To provide a performance comparison anyway, we implemented a benchmark consisting of a full matrix multiplication. We chose the benchmark since ML networks usually contain convolutions or matrix multiplications in their linear layers, matrix multiplications being the faster of the two operations. We thus compare to the harder case. We ran the benchmark on a GPU (Nvidia H100) and within Intel SGX (2x Intel Xeon Gold 6438Y+).

The runtime results are depicted in Figure 5. As expected, the enclave is the slowest option regardless of matrix size, while the GPU is the fastest. We can see that the speedup is there orders of magnitude regardless of the matrix size and ranges between $f = 1\,000$ and $f = 5\,000$. That means that we always have a speedup of over 1 000, meaning that $f > 1\,000$. In the provable secure scenario, this gives us a speedup of at least 2.2 up to 11.

5.2 Integrity Add-on

The client can check the FPU's honesty at three points: When $S = r_i$ is generated, when the unmasking values u_i are calculated and when the results of the linear layers are

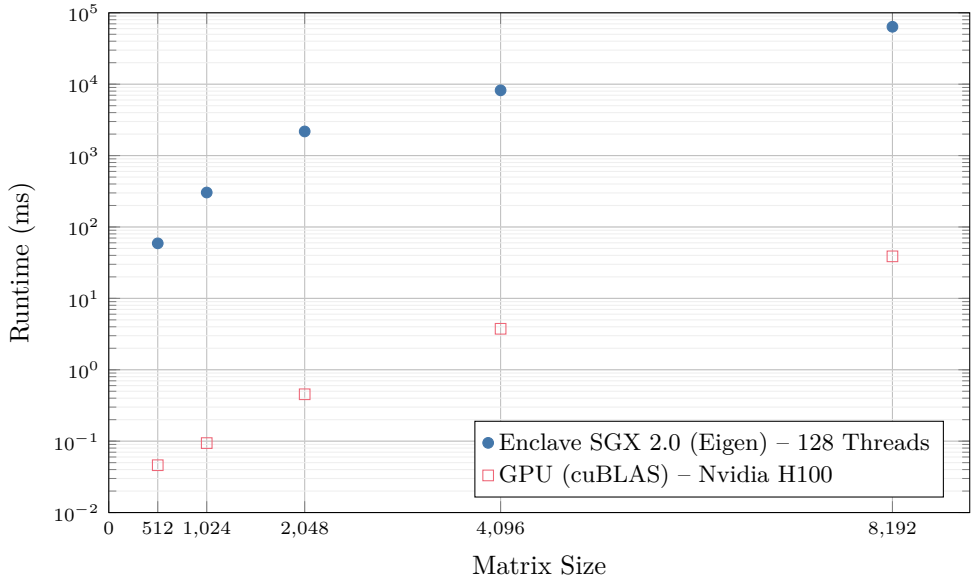

Figure 5: Runtime comparison for a matrix multiplication within the enclave and on the GPU

returned to the TEE. Slalom uses an error term of $t = 2^k$ and achieves t-integrity by k repetitions of Freivalds' algorithm per layer, as shown in the following theorem.

Theorem 4 (Theorem 3.2 in [TB19])**.** *Assume that all random values are generated using a secure PRNG with security parameter λ. Then Slalom provides t-integrity for $t = 2^k$ if Freivalds' algorithm is repeated k times per layer.*

Detect FPU Cheating During Set Generation The TEE can detect a dishonest FPU in this case if a PRNG with leap ahead property is used: The TEE can generate k random values from S itself and compare them to the ones sent by the FPU. The hardest case to detect would be the FPU cheating on exactly one generated value. Since n values are generated, the probability of not detecting the one the FPU cheated on when randomly reproducing k values is $\approx 1 - \frac{k}{n}$.

The detection probability obviously increases drastically if the FPU cheats more than once.

Detect FPU Sending Dishonest Unmasking Values Assuming that the FPU generated the masking values fairly, we can apply Freivalds' algorithm to check if it also calculated and sent honest unmasking values [Fre77]. This probabilistic test with one-sided error checks if two $j \times j$ matrices where multiplied correctly without comparing all the elements individually. To check, we aggregate n set elements from the masking set into an $n \times n$ matrix M and the corresponding n unmasking vectors into an $n \times n$ matrix U. To check if the multiplication

$$W \cdot M = U$$

was performed honestly, we sample a random binary vector t of length n and calculate

$$W \cdot (Mt)) - Ut = 0.$$

If we repeat this k times, the error term is bound by $\mathsf{e} \leq \left(\frac{1}{2^{2k}}\right)$ since we use binary vectors. To achieve the desired error term of 2^{-40}, we need $k \geq 20$.

Detect FPU Cheating During Inference Detecting if the FPU sent an honest inference result is similar to detecting if it cheated in creating the unmasking values. Here we build batches from the masked input values instead of the set elements, and compare with the inference results. We still use binary vectors for t, so to achieve an error term $\mathsf{e} \leq 2^{-40}$ similar to Slalom, we also require at least 20 repetitions.

As opposed to Slalom, we only use random numbers from $\{0,1\}^n$. This means we can perform Freivalds' check without relaxing our assumptions about the TEE: Since we use a binary vector to detect the error, we can still calculate everything using only additions and modular reductions, at the cost of having to perform about 10 times more tests.

Equipped with this integrity test, we can conclude our main theorem.

Theorem 5. *Let Carnival be the above protocol to compute* NN *with the integrity test. Assume that all random values are generated using a secure PRNG with security parameter λ and that the knapsack function $f = f_{p,S}$ is a one-way function chosen with security parameter λ. Then, Carnival is a secure outsourcing scheme for* NN *guaranteeing correctness, privacy, and t-integrity for $t = 2^k$ if Freivalds' algorithm is repeated k times per layer.*

5.3 Using Existing Elements as Set Items

The pseudo-random elements in the set can be generated from existing elements such as AES S-Boxes or program code in main memory. While these are not entirely random, [NPH18, VKV98] showed that they can be used to generate pseudo-random elements. By e.g., using the current software on IoT devices as randomness, the devices have a "build-in" set from the beginning and do not need to save or receive additional randomness. Even software updates are not a problem as long as they happen fairly simultaneously: Exchanging the set does not generate any more effort for the participants.

6 Related Work

Securing outsourced ML inference is a very active research area and many high-quality SoK papers have been published recently. We focus on direct follow-up works of Slalom and efforts to improve preprocessing. For a wider overview on this drastically growing field, we refer to the papers mentioned in Section 2.4 and to surveys such as [LXW+21, TCBK20, QHF+23, ELD24].

Many outsourced ML approaches divide their calculations into an offline preprocessing phase and an online phase. However, the preprocessing phase is often not included in the performance analysis [SBBE23, WWP22] or optimization efforts: Authors argue that it can be performed offline ahead of time and thus does not impact the total time required for the inference. An exception are the works Muse, Delphi, Gazelle and SIMC [LMSP21, MLS+20, JVC18, CGOS22], which all use additive HE for the linear layers and binary GCs for the non-linear layers. The bottleneck of this architecture is the conversion between the two, which requires extensive amounts of communication and calculation. The earliest work is Gazelle, which has roughly 60% of its runtime and 80% of its communication in the offline phase. With Delphi, Mishra et al. showed that they can push 99% of their computational load to the input independent preprocessing phase and also reduce the overall cost of a (deep) network by approximating the activation functions instead of using ReLUs. Muse builds on Delphi and focuses on a different security goal, leading to worse runtimes in both preprocessing and online phase. The HE and authenticated Beaver triples used in the non-linear layers of Muse replaced by oblivious transfer and onetime pad encryptions in SIMC, leading to a significant performance and communication gain in the preprocessing phase while maintaining the same values in the online phase.

It is worth mentioning that speeding up the preprocessing of outsourced computation is a field of research independent of machine learning, and efforts to enhance the performance and communication of preprocessing steps are of course present in SMPC tasks. Scholl et all. give a good overview on this topic [SSW17]. To the best of our knowledge, there has been no prior attempt to speedup the preprocessing phase of Slalom.

Slalom uses a combination of TEE and masking to process confidential values on an insecure GPU. It only supports inference, as the whole preprocessing phase relies on fixed model parameters, which are not given during training. With DarKnight, Hashemi et al. directly follow up on this issue of Slalom and add another wrapper to enable learning as well. They do however leave the existing framework as it is and do not alter the preprocessing phase [HWA22]. We thus expect that our performance improvements could also be used by DarKnight.

7 Conclusion

With S@C, we developed a novel way of calculating a linear function on random variables using the Subset sum problem over finite fields. S@C can be used in various scenarios. We decided to showcase the improvement of the offline phase of the Slalom framework, which generates masks for ML inputs by evaluating the linear layers on random matrices. S@C is a step towards more practical homomorphic encryption since it drastically reduces the computation time during offline preprocessing.

References

[BBG+20] Gilles Barthe, Sandrine Blazy, Benjamin Grégoire, Rémi Hutin, Vincent Laporte, David Pichardie, and Alix Trieu. Formal verification of a constant-time preserving C compiler. *Proc. ACM Program. Lang.*, 4(POPL):7:1–7:30, 2020. doi:10.1145/3371075.

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 633–666. Springer, 2020. doi:10.1007/978-3-030-64834-3_22.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-20465-4_11.

[BMD+17] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. In William Enck and Collin Mulliner, editors, *11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017*. USENIX Association, 2017. URL: https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser.

[BMW+18] 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 William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 991–1008. USENIX Association, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck.

[BPS17] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Sgx-step: A practical attack framework for precise enclave execution control. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP 2017, Shanghai, China, October 28, 2017*, pages 4:1–4:6. ACM, 2017. doi:10.1145/3152701.3152706.

[Bri84] Ernest F Brickell. Solving low density knapsacks. In *Advances in cryptology*, pages 25–37. Springer, 1984. doi:10.1007/978-1-4684-4730-9_2.

[CB19] Joseph I. Choi and Kevin R. B. Butler. Secure multiparty computation and trusted hardware: Examining adoption challenges and opportunities. *Secur. Commun. Networks*, 2019:1368905:1–1368905:28, 2019. doi:10.1155/2019/1368905.

[CBL+18] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *CoRR*, abs/1811.09953, 2018. URL: http://arxiv.org/abs/1811.09953, arXiv:1811.09953.

[CD16] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016. URL: http://eprint.iacr.org/2016/086.

[CGOS22] Nishanth Chandran, Divya Gupta, Sai Lakshmi Bhavana Obbattu, and Akash Shah. SIMC: ML inference secure against malicious clients at semi-honest cost. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 1361–1378. USENIX Association, 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/chandran.

[CJM20] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 189–218. Springer, 2020. doi:10.1007/978-3-030-56877-1_7.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017. doi:10.1007/978-3-319-70694-8_15.

[CLD16] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 857–874. USENIX Association, 2016. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan.

[CPTH21] Sylvain Chatel, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Sok: Privacy-preserving collaborative tree-based model learning. *Proc. Priv. Enhancing Technol.*, 2021(3):182–203, 2021. URL: https://doi.org/10.2478/popets-2021-0043, doi:10.2478/POPETS-2021-0043.

[CR88] Benny Chor and Ronald L Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5):901–909, 1988. doi:10.1109/18.21214.

[DFH+24] Abdulrahman Diaa, Lucas Fenaux, Thomas Humphries, Marian Dietz, Faezeh Ebrahimianghazani, Bailey Kacsmar, Xinda Li, Nils Lukas, Rasoul Akhavan Mahdavi, Simon Oya, Ehsan Amjadian, and Florian Kerschbaum. Fast and private inference of deep neural networks by co-designing activation functions. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024 (in Print)*. USENIX Association, 2024. URL: https://www.usenix.org/system/files/sec24summer-prepub-373-diaa.pdf.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012. doi:10.1007/978-3-642-32009-5_38.

[ELD24] Soumia Zohra El Mestari, Gabriele Lenzini, and Huseyin Demirci. Preserving data privacy in machine learning systems. *Computers & Security*, 137:103605, 2024. URL: https://www.sciencedirect.com/science/article/pii/S0167404823005151, doi:10.1016/j.cose.2023.103605.

[Fre77] Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP congress*, volume 839, page 842, 1977. URL: https://api.semanticscholar.org/CorpusID:45405368.

[GAGN15] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1737–1746. JMLR.org, 2015. URL: http://proceedings.mlr.press/v37/gupta15.html.

[GDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016. URL: http://proceedings.mlr.press/v48/gilad-bachrach16.html.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 25–32. ACM, 1989. doi:10.1145/73007.73010.

[HMSY21] Aditya Hegde, Helen Möllering, Thomas Schneider, and Hossein Yalame. Sok: Efficient privacy-preserving clustering. *Proc. Priv. Enhancing Technol.*,

2021(4):225–248, 2021. URL: https://doi.org/10.2478/popets-2021-0068, doi:10.2478/POPETS-2021-0068.

[HWA22] Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. Darknight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware. *CoRR*, abs/2207.00083, 2022. URL: https://doi.org/10.48550/arXiv.2207.00083, arXiv:2207.00083, doi:10.48550/ARXIV.2207.00083.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of cryptology*, 9(4):199–216, 1996. doi:10.1007/BF00189260.

[Int21] Intel. Intel software guard extensions developer reference for linux* os, 2021. URL: https://download.01.org/intel-sgx/sgx-linux/2.14/docs/Intel_SGX_Developer_Reference_Linux_2.14_Open_Source.pdf.

[Int24] Intel. Intel Trust Domain Extensions (Intel TDX) Module Base Architecture Specification, March 2024. Revision 348549-004US.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1651–1669. USENIX Association, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar.

[Kap16] David Kaplan. AMD x86 memory encryption technologies. Talk, USENIX 2016, August 2016.

[KG11] Aniket Kate and Ian Goldberg. Generalizing cryptosystems based on the subset sum problem. *International Journal of Information Security*, 10(3):189–199, 2011. URL: https://doi.org/10.1007/s10207-011-0129-2, doi:10.1007/S10207-011-0129-2.

[KRC+20] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 336–353. IEEE, 2020. doi:10.1109/SP40000.2020.00092.

[Lem79] Abraham Lempel. Cryptology in transition. *ACM Computing Surveys (CSUR)*, 11(4):285–303, 1979. doi:10.1145/356789.356792.

[LFFJ20] Qian Lou, Bo Feng, Geoffrey Charles Fox, and Lei Jiang. Glyph: Fast and accurately training deep neural networks on encrypted data. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/685ac8cadc1be5ac98da9556bc1c8d9e-Abstract.html.

[LKS+20] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: an open framework for architecting trusted execution environments. In Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan

Kostic, and Margo I. Seltzer, editors, *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, pages 38:1–38:16. ACM, 2020. doi:10.1145/3342195.3387532.

[LLD+22] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. Design and verification of the arm confidential compute architecture. In Marcos K. Aguilera and Hakim Weatherspoon, editors, *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 465–484. USENIX Association, 2022. URL: https://www.usenix.org/conference/osdi22/presentation/li.

[LMSP21] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure inference resilient to malicious clients. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2201–2218. USENIX Association, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/lehmkuhl.

[LO85] Jeffrey C Lagarias and Andrew M Odlyzko. Solving low-density subset sum problems. *Journal of the ACM (JACM)*, 32(1):229–246, 1985. doi:10.1145/2455.2461.

[LXW+21] Ximeng Liu, Lehui Xie, Yaopeng Wang, Jian Zou, Jinbo Xiong, Zuobin Ying, and Athanasios V. Vasilakos. Privacy and security issues in deep learning: A survey. *IEEE Access*, 9:4566–4593, 2021. doi:10.1109/ACCESS.2020.3045078.

[MAB+13] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In Ruby B. Lee and Weidong Shi, editors, *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013*, page 10. ACM, 2013. doi:10.1145/2487726.2488368.

[MH78] Ralph Merkle and Martin Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE transactions on Information Theory*, 24(5):525–530, 1978. doi:10.1109/TIT.1978.1055927.

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Comput. Complex.*, 16(4):365–411, 2007. doi:10.1007/s00037-007-0234-9.

[MIE17] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. Cachezoom: How SGX amplifies the power of cache attacks. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 69–90. Springer, 2017. doi:10.1007/978-3-319-66787-4_4.

[MLS+20] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2505–2522. USENIX Association, 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/mishra.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 465–484. Springer, 2011. doi:10.1007/978-3-642-22792-9_26.

[MOG+20] Kit Murdock, David F. Oswald, Flavio D. Garcia, Jo Van Bulck, Frank Piessens, and Daniel Gruss. Plundervolt: How a little bit of undervolting can create a lot of trouble. *IEEE Secur. Priv.*, 18(5):28–37, 2020. doi:10.1109/MSEC.2020.2990495.

[MR18] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018. doi:10.1145/3243734.3243760.

[MSK+20] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. Darknetz: towards model privacy at the edge using trusted execution environments. In Eyal de Lara, Iqbal Mohomed, Jason Nieh, and Elizabeth M. Belding, editors, *MobiSys '20: The 18th Annual International Conference on Mobile Systems, Applications, and Services, Toronto, Ontario, Canada, June 15-19, 2020*, pages 161–174. ACM, 2020. doi:10.1145/3386901.3388946.

[NAA22] Yue Niu, Ramy E. Ali, and Salman Avestimehr. 3legrace: Privacy-preserving DNN training over tees and gpus. *Proc. Priv. Enhancing Technol.*, 2022(4):183–203, 2022. URL: https://doi.org/10.56553/popets-2022-0105, doi:10.56553/POPETS-2022-0105.

[NC23] Lucien K. L. Ng and Sherman S. M. Chow. Sok: Cryptographic neural-network computation. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 497–514. IEEE, 2023. doi:10.1109/SP46215.2023.10179483.

[NLDD23] Deepika Natarajan, Andrew D. Loveless, Wei Dai, and Ronald G. Dreslinski. Chex-mix: Combining homomorphic encryption with trusted execution environments for oblivious inference in the cloud. In *8th IEEE European Symposium on Security and Privacy, EuroS&P 2023, Delft, Netherlands, July 3-7, 2023*, pages 73–91. IEEE, 2023. URL: https://doi.org/10.1109/EuroSP57164.2023.00014, doi:10.1109/EUROSP57164.2023.00014.

[NPH18] Florian Neugebauer, Ilia Polian, and John P. Hayes. S-box-based random number generation for stochastic computing. *Microprocess. Microsystems*, 61:316–326, 2018. URL: https://doi.org/10.1016/j.micpro.2018.06.009, doi:10.1016/J.MICPRO.2018.06.009.

[Odl90] Andrew M Odlyzko. The rise and fall of knapsack cryptosystems. In *Symposia of Applied Mathematics*, pages 75–88, 1990. URL: https://www-users.cse.umn.edu/~odlyzko/doc/arch/knapsack.survey.pdf.

[OT04] Keiji Omura and Keisuke Tanaka. Density attack to the knapsack cryptosystems with enumerative source encoding. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 87(6):1564–1569, 2004.

[Pis05] David Pisinger. Where are the hard knapsack problems? *Comput. Oper. Res.*, 32:2271–2284, 2005. doi:10.1016/j.cor.2004.03.002.

[PMG+17] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 506–519. ACM, 2017. doi:10.1145/3052973.3053009.

[QHF+23] Hong Qin, Debiao He, Qi Feng, Muhammad Khurram Khan, Min Luo, and Kim-Kwang Raymond Choo. Cryptographic primitives in privacy-preserving machine learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–17, 2023. doi:10.1109/TKDE.2023.3321803.

[RRK18] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 2:1–2:6. ACM, 2018. doi:10.1145/3195970.3196023.

[SBBE23] Jonas Sander, Sebastian Berndt, Ida Bruhns, and Thomas Eisenbarth. DASH: accelerating distributed private machine learning inference with arithmetic garbled circuits. *CoRR*, abs/2302.06361, 2023. URL: https://doi.org/10.48550/arXiv.2302.06361, arXiv:2302.06361, doi:10.48550/ARXIV.2302.06361.

[Sha82] Adi Shamir. A polynomial time algorithm for breaking the basic merkle-hellman cryptosystem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 145–152. IEEE, 1982. doi:10.1109/SFCS.1982.5.

[SMF21] Muhammad Usama Sardar, Saidgani Musaev, and Christof Fetzer. Demystifying attestation in intel trust domain extensions via formal verification. *IEEE Access*, 9:83067–83079, 2021. doi:10.1109/ACCESS.2021.3087421.

[SSW17] Peter Scholl, Nigel P. Smart, and Tim Wood. When it's all just too much: Outsourcing mpc-preprocessing. In Máire O'Neill, editor, *Cryptography and Coding - 16th IMA International Conference, IMACC 2017, Oxford, UK, December 12-14, 2017, Proceedings*, volume 10655 of *Lecture Notes in Computer Science*, pages 77–99. Springer, 2017. doi:10.1007/978-3-319-71045-7_4.

[TB19] Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=rJVorjCcKQ.

[TCBK20] Harry Chandra Tanuwidjaja, Rakyong Choi, Seunggeun Baek, and Kwangjo Kim. Privacy-preserving deep learning on machine learning as a service - a comprehensive survey. *IEEE Access*, 8:167425–167447, 2020. doi:10.1109/ACCESS.2020.3023084.

[Tra19] Florian Tramer. slalom. https://github.com/ftramer/slalom.git, 2019.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 601–618. USENIX Association, 2016. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer.

[VKV98] Karthik Visweswariah, Sanjeev R. Kulkarni, and Sergio Verdú. Source codes as random number generators. *IEEE Trans. Inf. Theory*, 44(2):462–471, 1998. doi:10.1109/18.661497.

[WMW23] Qizheng Wang, Wenping Ma, and Weiwei Wang. B-LNN: inference-time linear model for secure neural network inference. *Inf. Sci.*, 638:118966, 2023. URL: https://doi.org/10.1016/j.ins.2023.118966, doi:10.1016/J.INS.2023.118966.

[WRPE24] Jan Wichelmann, Anja Rabich, Anna Pätschke, and Thomas Eisenbarth. Obelix: Mitigating side-channels through dynamic obfuscation. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 4182–4199. IEEE, 2024. doi:10.1109/SP54263.2024.00261.

[WWP22] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A GPU platform for secure computation. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 827–844. USENIX Association, 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/watson.

[WZB+23] Qifan Wang, Lei Zhou, Jianli Bai, Yun Sing Koh, Shujie Cui, and Giovanni Russello. *HT2ML*: An efficient hybrid framework for privacy-preserving machine learning using HE and TEE. *Comput. Secur.*, 135:103509, 2023. URL: https://doi.org/10.1016/j.cose.2023.103509, doi:10.1016/J.COSE.2023.103509.

[XCP15] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 640–656. IEEE Computer Society, 2015. doi:10.1109/SP.2015.45.

[ZLT+24] Guangsheng Zhang, Bo Liu, Huan Tian, Tianqing Zhu, Ming Ding, and Wanlei Zhou. How does a deep learning model architecture impact its privacy? In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024 (in Print)*. USENIX Association, 2024. URL: https://www.usenix.org/system/files/sec24summer-prepub-365-zhang-guangsheng.pdf.