Towards Private Deep Learning-based Side-Channel Analysis using Homomorphic Encryption

Fabian Schmid[1], Shibam Mukherjee[1,5], Stjepan Picek[2], Marc Stöttinger[3], Fabrizio De Santis[4], Christian Rechberger[1]

[1] Graz University of Technology, Austria
[2] Radboud University, The Netherlands
[3] RheinMain University of Applied Science , Germany
[4] Siemens AG, Germany
[5] Know-Center GmbH, Austria

Abstract. Side-channel analysis certification is a process designed to certify the resilience of cryptographic hardware and software implementations against side-channel attacks. In certain cases, third-party evaluations by external companies or departments are necessary due to limited budget, time, or even expertise with the penalty of a significant exchange of sensitive information during the evaluation process. In this work, we investigate the potential of Homomorphic Encryption (HE) in performing side-channel analysis on HE-encrypted measurements. With HE applied to side-channel analysis (SCA), a third party can perform SCA on encrypted measurement data and provide the outcome of the analysis without gaining insights about the actual cryptographic implementation under test. To this end, we evaluate its feasibility by analyzing the impact of AI-based side-channel analysis using HE (private SCA) on accuracy and execution time and compare the results with an ordinary AI-based side-channel analysis (plain SCA). Our work suggests that both unprotected and protected cryptographic implementations can be successfully attacked already today with standard server equipment and modern HE protocols/libraries, while the traces are HE-encrypted.

Keywords: Side-channel Analysis · Deep Learning · Neural Networks · Homomorphic Encryption · Private AI

1 Introduction

Cloud services have become increasingly popular in recent years, as they provide organizations and individuals with the ability to store, process, and analyze a large amount of data without investing in expensive hardware and infrastructure. One prominent example of technologies used in the cloud is artificial intelligence and machine learning, which enable organizations to extract valuable insights from large datasets, leading to improved decision-making in many fields of technology and business. However, using cloud services also raises concerns about privacy violations such as data breaches, unauthorized access, data misuse, lack of transparency, and compliance issues, e.g., GDPR. In general, these kinds of privacy issues arise if any third party (not necessarily a cloud service) is required to operate on data that contains sensitive information that is supposed not to be leaked or exposed. Generally, there are different reasons for outsourcing data and computation to a third party like a cloud service like lack of computational power (e.g., computationally expensive tasks have to be performed), lack of expertise (e.g., a third-party service is used

to perform ad-hoc computational tasks for which specific expertise is needed), or costs effectiveness (e.g., it may be more convenient to perform some tasks in the cloud due to associated maintenance expenses).

Side-channel analysis dates back to the seminal work of Kocher [KJJ99] and represents today one of the pillars for security evaluations of information security products. In particular, profiling side-channel attacks represent a de-facto standard evaluation technique for evaluation, and deep learning is particularly powerful in this task [PPM+23].

In the context of side-channel analysis, the reasons for outsourcing side-channel analysis to a third-party (e.g., cloud provider) can be traced back to a lack of infrastructure to perform side-channel evaluation in-house due to an availability issue, or the lack of time or resources to stay up-to-date with the theory and implementation of state-of-the-art side-channel attacks (e.g., it may be more convenient to outsource the analysis of side channels to a specialized third party to reduce the time and effort required to perform the analysis internally and train the staff.) In particular, an independent third-party assessment of side-channel analysis is an important step in the certification process, which aims to certify the resilience of cryptographic hardware and software implementations against side-channel attacks. This process typically involves long cycles of evaluations, sometimes including third parties, to reach a level of confidence that an implementation will pass security certification. For this reason, evaluations are often carried out in advance by independent third parties (internal or external) to reduce the costly evaluation cycles during the certification process. To date, the interaction with a third party happens with a noticeable exchange of proprietary information between the parties – in the form of a "prepared" device under test and/or a set of traces collected under precise assumptions.

Homomorphic encryption (HE) is a cryptographic primitive that enables computation on encrypted data without the need to decrypt it first. A third-party evaluator can privately interact with a customer desiring a side-channel security evaluation for a device under test using HE. The evaluator receives the HE traces from the customer, analyses them in encrypted form homomorphically, and reports for any possible leakage to the customer without learning anything about the traces or the final results. Once the customer is confident about the security of its device, it can send it to the certifying authority for final security certification.

With HE, a third-party provider can analyze the traces and provide the outcome of the analysis without gaining insights into the underlying cryptographic implementation and its leakage profile. At the same time, the process of pre-certification and its costs can be amortized across more trials executed through a cloud service where homomorphically encrypted traces can be uploaded to the cloud for preliminary testing.

Related work

Artificial intelligence techniques have recently gained significant attention from the side-channel community, owing to their success in various domains where data analysis is used. In particular, the applicability of deep learning to side-channel analysis has spurred numerous investigations into its performance and effectiveness, showing excellent results. For instance, deep neural networks (DNNs) outperformed traditional profiling attacks such as template attacks in practice [CRR03, RO04] and were proven highly successful even on devices protected with various countermeasures. Machine learning has been used in SCA for more than two decades already. In the first few years, the focus was on simpler machine learning techniques like naive Bayes [PHG17], random forest [LMBM13], and support vector machines [HZ12]. Besides evaluating the attack performance (i.e., how many traces is required to break the target), an important direction was to conduct efficient feature engineering [PHJB19]. In 2016, Maghrebi et al. showcased how deep learning (deep neural networks) can break various targets [MPP16]. This work started a series of research on deep learning-based SCA (DLSCA), where most of the works explored how to conduct as

efficient as possible attacks. For instance, Cagli et al. demonstrated how data augmentation improves the performance of attacks on desynchronized datasets [CDP17]. Kim et al. showed how convolutional neural networks (CNNs) can break various targets and that regularization reduces overfitting, further improving the attack performances [KPH+19]. Perin et al. demonstrated how ensembles of neural networks can outperform a single neural network [PCP20]. Zaid et al. provided the first methodology to build convolutional neural networks for SCA [ZBHV19]. Rijsdijk et al. [RWPP21] and Wu et al. [WPP22] discussed how to improve the hyperparameter tuning phase with reinforcement learning and Bayesian optimization, respectively. Finally, more recent results by Lu et al. [LZC+21], and Perin et al. [PWP22] demonstrated that deep neural networks work better if no prior feature selection is done. These works showcase that it is possible to break protected targets even with a single attack measurement.

HE has already been successfully tested in various real-world applications, including energy forecasting predictions [BCIV17], healthcare [WJT+17, BLN14, BBH+22], and financial services [MHS+19], showing that it is a viable solution for safeguarding privacy of AI computations. Regarding classification based on homomorphic encryption, there are many works in the literature [GDL+16, DSC+19, DKS+19, CGL+20, CBL+18, CJP21, JVC18, VJH21]. These works brought many improvements and dedicated optimizations. However, they all face the trade-off between limited neural network depth and computational overhead. While this limitation limits the use cases for private ML, these works show the practicability of applying privacy-preserving methods to less complex neural networks.

Contributions

The main goal of this work is to investigate the technical feasibility of HE-assisted side-channel analysis using convolutional neural networks to highlight strengths and understand limitations in terms of performance and accuracy. This work shows how to perform private neural network-based side-channel analysis using fully homomorphic encryption techniques. First, we evaluate two datasets, including both unprotected and protected software implementations of AES-128, showing that both HE-encrypted datasets can be successfully attacked with standard server equipment and modern HE protocols/libraries. Then, we discuss the strengths and limitations of HE-assisted DLSCA: on the positive side, we show the technical feasibility of HE-assisted DLSCA both in terms of accuracy and execution time. On the negative side, our system works in the semi-honest trust model where all the parties shall never actively deviate from the HE protocol in order to ensure the validity of the results.

Please note that this is the first attempt in the open literature to evaluate the feasibility of HE-assisted side-channel analysis using deep learning neural networks with the goal of establishing the basis and paving the way for future studies and advancements in this emerging area of HE in regard to side-channel analysis.

The corresponding implementation is available on request and will be made public with the paper.

Organization

The rest of this paper is organized as follows. In Section 2, background information on HE and DL-based SCA is provided. In Section 3, a description of how to perform HE-assisted DL-based SCA is given. Section 4 overviews the setup and conducted experiments. Section 5 provides a comprehensive discussion about the obtained results. Conclusions and directions for future work are provided in Section 6.

2 Background

First, we provide basics related to homomorphic encryption to enable the reader to understand the fundamental operations done there. Then, we provide information related to side-channel analysis with neural networks.

2.1 Homomorphic Encryption (HE)

Cryptographic algorithms aim at protecting our data regarding confidentiality, integrity, and authenticity. Existing solutions shield our communication (data in transit) and data storage (data at rest). However, the emergence of cloud computing has faced cryptographic research with fundamentally new challenges and led to new research areas to secure data during computations. Before, there were only two options when storing data in the cloud was necessary. First, download and decrypt the data, perform computations, encrypt, and upload again. Second, leave the data unprotected in the cloud to use its services. These options are a trade-off between runtime and security. However, one often uses cloud services due to the need for more computational resources, such as fast hardware or significant storage capabilities. Furthermore, given use cases like machine learning as a Service (MlaaS), where pre-trained models are proprietary information, users are forced to entrust the companies with their data. This raises the need for a new privacy-preserving primitive.

Operations on Encrypted Data With homomorphic encryption (HE) we [RAD78, Gen09] refer to encryption schemes that allow operations on encrypted data that directly carry over to the underlying plaintext domain. Without decryption, a server can execute some algorithm on encrypted data without learning anything about the input or the result of the computation. Privacy-preserving solutions like HE have many applications, such as statistical evaluations [AEH15, LKS17], medical domain [LYS15, WZD+16], or financial domain [Bag19, SA21, SBBV22].

The idea of performing operations on encrypted data seems to have been mentioned in a paper for the first time in 1978 by Rivest, Adleman, and Dertouzous [RAD78] in terms of *privacy homomorphism*. Some (early) encryption schemes already support homomorphic encryption, but only *partially*, i.e., in a single operation. For example, ElGamal [Gam84] is multiplicatively homomorphic, while Pallier [Pai99] is additively homomorphic. Encryption schemes with both additive *and* multiplicative homomorphisms rely on noise to secure the secret. This inherent noise increases with the number of homomorphic operations and thus limits the maximum number of operations. We refer to such schemes as *Somewhat* HE (SHE) schemes. Allowing unlimited operations was an open problem until 2009 when Gentry introduced the first *Fully* HE (FHE) scheme [Gen09]. The key step to FHE is the bootstrapping procedure, allowing to reset the noise of a ciphertext for unbounded depth in the evaluation circuit.

Today, FHE schemes can be divided into binary [DM15, CGGI20] and arithmetic [Bra12, BGV12, FV12, CKKS17], depending on the types of operations supported. Binary FHE schemes have a bootstrapping step after each computation, naturally giving them an unbounded number of operations. However, encrypting binary values gives a massive ciphertext expansion and runtime penalty when working with integers. Arithmetic FHE schemes, on the other hand, provide several optimizations for computations on integers (in the case of BGV and BFV) or fixed precision numbers (in the case of CKKS). In contrast to binary schemes, bootstrapping is expensive. In general, one can boost the number of operations one can perform before the noise grows too large by increasing the parameters of HE. Multiplication is the primary driver of noise growth. In practice, one counts the required multiplications and tunes the parameters to fit the specific use case. We refer to such an instantiation as *Levelled* HE (LHE), as we can perform exactly "L"

levels of multiplications. Next to the noise problem, these LHE schemes only allow for homomorphic addition and multiplication in all settings, and vector rotations when using special encoding. For every non-linear function, such as typical activation functions in neural networks, it is most efficient to work with polynomial approximations like Taylor approximation or Chebyshev approximation.

When considering the HE scheme as a black box for secure computation, one could instantiate our use case with any of the described methods above. However, we do want to motivate our use of the CKKS scheme. The main driving factor of multiplicative depth in neural network inference tasks is the number of layers and degree of the approximation polynomials. Since our task can be achieved with only a few layers (see, e.g., [PWP22] where the authors required neural networks with only a few layers to break the targets protected even with masking and hiding countermeasures) and a low degree of the approximation polynomial gives us good results, the multiplicative depth is not an issue. Thus, we can select an arithmetic LHE scheme. Contrary to other LHE schemes, CKKS is built to work directly with approximate numbers. This property makes it a natural fit for neural network tasks. Finally, CKKS allows for batching of ciphertexts. This optimization allows encrypting multiple data points into a single ciphertext and performing operations in a *single instruction multiple data* (SIMD) fashion.

In summary, HE is a very active research domain with many developments lately. While the malleability of ciphertexts was initially an unwanted side effect, the homomorphic properties have become a hope for privacy-preserving algorithms. Given our use case, we evaluated the state-of-the-art HE schemes and weighed their trade-offs. At first, we compared the binary versus the arithmetic HE schemes. When dealing with approximate numbers (or integers) and the multiplicative depth of the encrypted computation is known a priori, the optimizations of LHE schemes outweigh their noise limitation. Finally, within the arithmetic schemes, only the CKKS scheme naturally supports approximate numbers, making it a perfect fit for our use case.

Homomorphic Encryption applied to Machine Learning In the domain of privacy-preserving machine learning, research focuses on applying privacy-preserving protocols to the training and classification of machine learning models. While HE generally allows encrypted computations, it is necessary to weigh the drawbacks and benefits of the different schemes carefully. The general properties of different HE schemes translate to their application in machine learning. Binary schemes have the advantage of unlimited circuit depth and can be applied to deep neural networks without adaptation [CGGI18]. However, working only on single-bit encryptions is a substantial performance penalty. Several works opt for the approximate arithmetic CKKS scheme [CKKS17]. The main issue there is the limited evaluation circuit. In CKKS, the multiplicative depth of the evaluation circuit depends on the scheme's parameters. More significant parameters incur runtime and memory costs while allowing for more operations. We will go further into the details of parameter choices in Section 2.1.1 and Section 3.2. In [BHM+19], the authors propose using more significant HE parameters to tackle the multiplicative depth of training a neural network on encrypted data. Given the multiplicative depth in deep neural network inference, Lee et al. [LKL+22] implement bootstrapping with significant HE parameters. Both approaches manage to solve their computation tasks but with high computational complexity. The first takes about five days to train a network for text classification, while the former takes around three hours for one classification in a 20-layer neural network. In [JVC18], the authors use additive HE combined with Secure multi-party Computation (MPC). They implement convolutional and dense neural network layers with an additive HE scheme while they evaluate the non-linear functions with MPC. This strategy allows them to compute complex neural networks while imposing high network and runtime costs on the client.

2.1.1 CKKS: Approximate Homomorphic Encryption

In this section, we will describe the specifics of the CKKS [CKKS17, CHK+18] cryptosystem. Fundamentally, CKKS provides an encoding strategy allowing us to encrypt complex vectors in integer-polynomial-based encryption schemes. While this approach can be instantiated with different methods, our discussion revolves around the BFV [Bra12, FV12] cryptosystem with CKKS encoding.

Encoding In BFV-like HE schemes, we operate on polynomial rings $R_q = \mathbb{Z}_q[X]/(X^n+1)$, for some large q and a power of two n. More precisely, public and private keys and ciphertexts are represented as integer polynomials with fixed degree n and coefficient modulus q. The individual polynomial coefficients are centered around zero. They lie within $(-q/2, q/2]$. It is important to note that CKKS provides a homomorphic encoding strategy to pack a vector of $\mathbb{C}^{\frac{n}{2}}$ complex numbers into a plaintext polynomial R_q. Additions and multiplications of an encoded plaintext polynomial lead to element-wise operations on the underlying vector of complex numbers. Next, to element-wise operations, this encoding strategy preserves the Galois automorphisms enabling vector rotations in the encrypted domain. Finally, the encoding procedure contains a rounding step since we map complex vectors to integer polynomials. CKKS multiplies the input numbers with the scale parameter Δ to avoid substantial accuracy loss in the encoding function. Typically, this scale is in a limited range $\Delta \in \{2^{20}, \ldots, 2^{60}\}$. The inherent noise required for the security of the encryption is then added in the least significant bits of the encoded numbers. Thus, increasing the Δ parameter decreases the noise impact. Ultimately, choosing the correct value for Δ is use-case specific and can be done empirically.

Encryption Scheme The CKKS encryption scheme is based on the *(Ring) Learning With Errors* Problem [Reg05, LPR10]. The fundamental assumption, a tuple of polynomials $a, a \cdot s + e \in R_q^2$ is indistinguishable from a random tuple $a, b \in R_q^2$. Here, $a \in R_q, s \in R_q$ are polynomials in our ring, and $e \in \chi$ is a noise term where χ is the bounded discrete Gaussian distribution. In BFV-like HE systems, reducing the norm of the secret polynomial s is necessary. Usually, without impact on the security assumptions, a ternary polynomial R_3 is sampled with coefficients in $\{-1, 0, 1\}$. In such HE systems, we set our secret key as $sk = s$ and our public key as $pk = (-a \cdot s + e, a) \in R_q^2$. The negation of the first public key polynomial simplifies notation. Given an encoded plaintext polynomial $m \in R_q$ and a public key $pk = (p_0, p_1)$, we can encrypt the underlying message by sampling $u \leftarrow R_3, e_0, e_1 \leftarrow \chi$ and setting $ct = (c_0, c_1) = (m + up_0 + e_0, up_1 + e_1)$. To decrypt, perform $m = [c_0 + c_1 s]_q$, where $[\cdot]_q$ indicates a modular reduction of the polynomial coefficients.

Homomorphic Operations In CKKS, homomorphic addition comes out of the box since $\mathsf{Enc}(\Delta m_1) + \mathsf{Enc}(\Delta m_2) = (c_{00}+c_{01}, c_{10}+c_{11}) = \mathsf{Enc}(\Delta(m_1+m_2))$. Plaintext multiplication, on the other hand, is equivalent to the tensor of the ciphertext $\mathsf{Enc}(\Delta m_1 \cdot \Delta m_2) = (c_{00} \cdot c_{01}, c_{00} \cdot c_{11} + c_{10} \cdot c_{01}, c_{10} \cdot c_{11}) \in R_q^3$. After performing a multiplication, we need to reduce the size of the ciphertext and adjust the scaling. To reduce the ciphertext back to two polynomials, we need a procedure called key switching, which we will discuss later. Next to the size expansion, the internal scaling, introduced by the encoding, is squared. It is necessary to rescale the ciphertext back to the old scaling factor, as keeping track of the scales of different ciphertexts is challenging. Further, after several multiplications, decryption would fail when the scale grows larger than the coefficient modulus. The rescaling procedure, thus, divides the ciphertext by the scale Δ.

Next to addition and multiplication, another vital operation is the homomorphic vector rotation. Applying Galois automorphisms on the ciphertexts leads to an underlying vector

rotation in the plaintext space. Again, such a transformation on the ciphertext requires a key-switching procedure with dedicated public keys.

In summary, we have the capabilities of SIMD addition, multiplication, and vector rotation. Before implementing an algorithm, we can make slight changes to make it more "HE-friendly". In practice, we often approximate non-linear functions with polynomials. Further, there are many adaptations to optimize numerical algorithms for batched computations [HS14, HS15, HS18]. When discussing our experiments, we will highlight the steps taken to make the algorithms more "HE-friendly".

Key Switching Both in the multiplication and rotation procedures, we arrive at ciphertexts that cannot be decrypted with the secret key s. A ciphertext product requires a squared secret key s^2, while after the homomorphic rotation $\psi_t(m)$, we require a transformed key $\psi_t(s)$, where t indicates the steps (left or right depending on the sign). While we could take that into account during decryption, we need to reset the ciphertext to the original key to allow for subsequent homomorphic operations. We can only add or multiply ciphertexts when tied to the same secret key. Enabling the server to perform this transformation requires a key switching key. This key-switching key $\texttt{ksk}_{s' \to s}$ can be considered an encryption of s' under s. Conceptually, in the case of relinearization after multiplication, the key has the form $\texttt{rlk} = \big([-(a \cdot s + e) + s^2]_q, a\big)$, where $a \leftarrow R_q, e \leftarrow \chi$. One can then relinearize the ciphertext (c_0, c_1, c_2) by computing $c_0' = [c_0 + \texttt{rlk}_0 \cdot c_2]_q$ and $c_1' = [c_1 + \texttt{rlk}_1 \cdot c_2]_q$ and set the ciphertext as (c_0', c_1'). In actual instantiations of this algorithm, there is a decomposition step for the key switching key to limit the noise growth of the operation.

RNS Decomposition The CKKS cryptosystem has a *residue number system* (RNS) variant [CHK+18]. As mentioned above, the main tuning parameters for the CKKS scheme are the polynomial degree n, the coefficient modulus q, and the scaling factor Δ. Increasing n boosts the security of the underlying lattice problem with a runtime penalty. The SIMD encoding of CKKS compensates for this penalty, as we can encode $n/2$ complex numbers. A large q gives us a higher precision and allows us to evaluate more levels of multiplications, while the operation on multi-precision integer coefficients is dramatically inefficient. The solution is to use the RNS decomposition to represent polynomials $\in R_q$ as multiple polynomials $\in R_{q_1} \times \cdots \times R_{q_L}$. All q_i are distinct primes that fit into single machine words and comprise the modulus $q = \prod_{i=1}^{L} q_i$. The L primes are roughly equivalent to the scale $q_i \approx \Delta, i \in \{1, \ldots, L\}$, such that $q \approx \Delta^L$. During the rescale operation applied after homomorphic multiplication, we divide both ciphertext polynomials by one modulus q_l and remove the RNS component. Moving from one level to the next $L \to L'$ we compute $q_{L'} \leftarrow \frac{q_L}{q_l}$ and $ct' \leftarrow [\frac{1}{q_l} \cdot ct]_{q_{L'}}$. This step introduces noise caused by the approximate relation $q_l \approx \Delta$, but it allows us to perform the complex rescaling operation with RNS decomposition. Since the modulus is divided by each rescaling operation, we are limited to L multiplications. Further, when faced with ciphertexts of different modulus sizes, one has to reduce the larger modulus before addition or multiplication is possible.

In our use case, we rely on the CKKS implementation SEAL [SEA23]. There, the total coefficient modulus is composed of two additional so-called special primes. These primes are set to size $\gamma + \log_2 \Delta$, where γ is the integer precision of the encrypted values in bits. We refer the reader to the literature [CKKS17, CHK+18, SEA23] for a comprehensive overview of the impact of RNS decomposition on the algorithms of CKKS.

2.2 Neural Network-based Side-channel Analysis

Side-channel analysis represents an attack method that targets the implementation rather than the cryptographic algorithm. Among SCA techniques, profiling side-channel at-

tacks are considered the most powerful because they make worst-case security assumptions [SKS09, PHPG21]. In these attacks, the attacker has access to a copy of a device that is used to build a profile, which can be then used to obtain the secret information from the target device [CRR03, MOP07].

Recently, the SCA community has focused its attention on deep learning techniques due to their success in various domains. The potential applicability of deep learning to SCA has triggered investigations into its performance. The results are promising: deep learning outperforms traditional profiling attacks attacks [CRR03, RO04, KSS10]. Remarkably, CNNs were successful even on devices protected with various countermeasures, as discussed previously.

Profiling side-channel analysis happens in two phases: profiling and attack. Let $\mathcal{X}^{tr} \in \mathbb{X}^{N \times P}$ be a 2D array (training input dataset) with N rows (side-channel traces) and P columns (point-of-interest (PoI)), where $\mathbb{X}$ represents the set of available side-channel traces. Let $\mathcal{Y}^{tr} \in \mathbb{Y}^{N}$ be an array of N rows (training labels), where $\mathbb{Y}$ contains the set of all possible classification labels, such that there exists a function $f : \mathcal{X}^{tr} \mapsto \mathcal{Y}^{tr}$. The goal of the profiling phase is to find the parameters θ $g_\theta(\mathcal{X}^{tr})$ that maximize the chance of outputting the expected value $\mathcal{Y}^{tr}$. In the attack phase, the goal is to predict the labels $\mathcal{Y}^{pr} \in \mathbb{Y}^{N}$ based on the attack traces $\mathcal{P}^{pr} \in \mathbb{P}^{Q \times C}$ and the trained model g_θ, where $\mathbb{P} \in [0, 1]$ (probability), Q is the number of attack traces and C is the number of labels (that depend on the cryptographic algorithm and the leakage model). Here, each value $\mathbf{p_i} \in \mathcal{P}^{pr}$, denotes the probability of obtaining $\mathbf{y_i} \in \mathcal{Y}^{pr}$ for a specific key $\mathbf{k}$ and some input $\mathbf{a} \in \mathcal{M}$, where $\mathcal{M}$ is the message space.

Finally, the cumulative sum $S(\mathbf{k})$ for any key byte candidate $\mathbf{k}$ is a side-channel distinguisher with a common maximum log-likelihood principle: $S(\mathbf{k}) = \sum_{i=1}^{Q} \log(\mathbf{p}_{i,\mathbf{y}})$. The cumulative sums for each possible key value from a key guessing vector are ordered per the probability of the key being correct (the first position in the vector is the most likely key, and the last position is the least likely key). The position of the correct key is called the key rank, and it denotes the remaining effort required by the attacker to break the target. To reduce the effects of a random choice of test measurements, it is common to assess the average behavior over many randomly selected traces, called guessing entropy (i.e., average key rank) [SMY09]. Besides considering the attack performance, one often also considers the complexity of the obtained models by assessing the number of trainable parameters [PCP20].

Commonly used neural networks in the profiling SCA are multilayer perceptron and convolutional neural networks [LBBH98]. Both neural networks are feed-forward neural networks, but CNNs consist of more types of layers - convolutional layer, pooling layer, and fully connected layer - while MLP consists of fully connected layers only. While both neural network types work well when considering SCA, CNNs have the potential advantage of being shift invariant, giving them more advantage when dealing with desynchronization countermeasure [CDP17].

3 HE-assisted DL-based Side-channel Analysis

This work applies the HE neural network classification of encrypted queries approaches from [JVC18, BHM+19, WSH+22] to SCA. We base our implementation on the homomorphic encryption library SEAL [SEA23] and the neural network inference implementation of CryptoTL [WSH+22]. While the latter primarily targets transfer learning, we adapted their implementations of the neural network layers and approximated non-linear functions to our use case. We extended the implementation to a more flexible setup that allows us to combine neural network layers more freely for different setups. In the following section, we give insights into the general setup of our HE-assisted DL-based SCA and highlight the challenges of making models “HE-friendly”.

On a high level, our design is divided into two sections: plain training and HE classification. First, we perform the profiling phase of the device under test (DUT) with access to corresponding key and plaintext data with measured traces. Given this data, the server trains a network state-of-the-art machine learning libraries. We will elaborate on the implementation specifics in Section 4.1. In the HE classification phase, we have a client with access to only a few traces. This client encrypts the relevant traces and sends them to the server. Given the encrypted traces and the pre-trained model, the server homomorphically classifies the query and returns the encrypted result. At no point does the server have access to the query or intermediate results. Ultimately, the client decrypts the classification result and updates its key prediction. Based on the results, the client analyses and improves the required protection of its cryptographic implementation on the target device. The same machine learning model has to be used during profiling and classification based on the neural network. Considering the limited operations we can perform on CKKS ciphertexts, designing these models in a "HE-friendly" way is essential. When discussing both phases of our setup, we will discuss relevant steps taken to adopt "HE-friendliness".

3.1 Profiling Phase

In the profiling phase (or training phase), the server has physical access to the DUT. First, the server measures traces $\mathcal{X}^{tr} \in \mathbb{X}^{N \times P}$ of the device and stores the corresponding key and input data. Before using the traces for profiling or classifying, as a preprocessing phase, we need to select points of interest, i.e., reduce the columns of the traces matrix to contain relevant data (PoIs). These points are the measurements that correlate most strongly with the key or a derived value. In case no statistical analysis is feasible, we work with the PoIs of the underlying dataset. The following sections will refer to the number of selected PoIs as P.

Given a dataset with statistical leakage, we select the most relevant PoIs by choosing the top K features using the Chi-Square test as the score function. The Chi-Square test is a statistical hypothesis test used to measure how well a model compares to the observed data. It is expressed as:

$$\chi^2 = \sum_{i=1}^{C} \frac{(x_i - m_i)^2}{m_i}, \tag{1}$$

where x_i is the observed value, m_i is the expected value, and C is the number of mutually exclusive classes. If there are N observations in the full dataset, then the expected value $m_i = N \times p_i$, where p_i is the hypothesized probability that an observation falls into the i^{th} class. A larger χ^2 value affirms that certain PoI might not provide valuable training information and thus can be discarded. Since this analysis is performed on plain data, the server has to inform the client about the selected PoIs. In our setting, we apply this method to test the impact of the feature count on the performance of our implementation. As a case study, we attack the first round of AES. We use either an unprotected implementation or measurements recorded with the masking countermeasure enabled. Thus, we aim to detect correlations between our traces and the targeted intermediate values. Generally, our attacks only target individual key bytes. Given the key k and the plaintext p as an array of 16 bytes, we attack the i^{th} byte by targeting the output of the AES S-box. More specifically, we target the intermediate result of S-box in the first round of AES. Given the traces with reduced columns and the labels from the associated data, we train a deep learning classifier. Depending on the computational capabilities of the server, we need to fine-tune the network parameters with particular regard to the HE restrictions. While we have observed that evaluating deep neural networks with CKKS is possible [LKL+22], we focus on smaller networks with a minor runtime penalty. However, deeper networks could be more suitable when attacking devices with stronger countermeasures [ZBHV19].

In our implementation, we present the feasibility of such a setting with both convolutional neural networks and multilayer perceptron. The details about the architectures are given later, but here, we note that we train with the ReLU activation function for hidden layers and the Softmax activation function for the output layer. As mentioned, we only have element-wise addition, multiplication, and rotation in the ciphertext domain. Thus, we must adapt our activation functions to be more "HE-friendly". The ReLU function can be approximated with polynomial approximation, while the softmax function is evaluated in plain.

First, we applied the Chebychev approximation to approach the ReLU function, which provided fast convergence. More precisely, we had satisfying results with a degree three approximation defined as:

$$ReLU_{approx}(x) = -0.0061728x^3 + 0.092593x^2 + 0.59259x + 0.49383. \tag{2}$$

The softmax activation function is evaluated in plain during training and classification. We will elaborate further on this in the respective section.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, i \in \{1, \ldots, K\}. \tag{3}$$

Softmax applies the standard exponentiation on each element x_i and normalizes these values into a probability distribution that sums up to 1. Finally, the output is passed through a categorical cross-entropy loss function to convert the probabilities to one-hot encoded classifications. We use the Adam optimization algorithm from the standard Keras package for training. Once the model is trained, the relevant model parameters are saved for later use by the server for SCA classification with HE.

3.2 Private Classification Phase

The classification procedure has the general setup of MlaaS. Initially, the client and the server agree on CKKS parameters that support the trained network. For our implementation, we give the following considerations on the choice of parameters. We need to dimension the parameters $\{n, q, \Delta, \gamma\}$. See Section 2.1.1 for an overview of CKKS and its parameters. For Δ, γ, the fixed-point scale, and the integer scale, it is sufficient to evaluate the correct setting empirically. In practice, depending on the multiplicative depth of the evaluation circuit (depending on the network), we scale our coefficient modulus to $\log_2 q = L \cdot \log_2 \Delta + 2(\log_2 \Delta + \gamma)$. In other words, we have L primes $q_i \approx \Delta$, and two distinct special primes $\log_2 q_s = \log_2 \Delta + \gamma$. Next, we set n according to either the recommendations of the homomorphic encryption standard [ACC+19] or the lattice hardness estimator [APS15]. For different levels of security, they provide the limit of q for a given n. Remember that a larger q has a negative impact on security and must be compensated with a larger n. Depending on the respective multiplicative depth and the resulting bit size in q, we set our n to the minimum required value with 128-bit security. Given our models, we work with $n = 16\,384$. Increasing security to 256-bit is achieved by doubling n, which roughly translates to doubling the runtime of HE computations. This level lies well in the NIST recommendation with a minimum of 80-bit security [Bar16].

Once the CKKS parameters are initialized, the server loads the model parameters and encodes them into the appropriate CKKS plaintext format. At the same time, the client prepares its query. The preprocessing must be the same in the attack and profiling phases. The server can let the client know the selected PoI or extract the relevant indices homomorphically. Since preprocessing, as described above, only works on unprotected data, we consider a server that shares this information with the client. Given protected datasets, the client sends the entire data to the server, and the feature extraction layers cover that part. When starting the classification phase, the server has an encrypted trace

and a plain model. The client sends multiple ciphertexts if the number of PoIs exceeds the encoding slots $P > n/2$. See Section 2.1.1 for more details on encoding and its limits. Given the encrypted trace, the server homomorphically evaluates the neural network. In the following, we discuss the challenges for the different neural network layers during classification.

Dense layer In all our networks, we employ fully connected layers, also called dense layers. These layers are performed as a matrix-vector multiplication. Halevi and Shoup [HS14, HS15, HS18] present the diagonal method and an optimized version with the baby-step-giant-step (BSGS) method. This strategy allows for the efficient implementation of plain-matrix, encrypted-vector multiplication. A square matrix is a precondition. Thus, we pad asymmetric matrices with zeroes in the shorter dimension. Given the square weight-matrix W with size t and the encrypted trace vector x, the diagonal method calculates

$$W\mathbf{x} = \sum_{i=0}^{t-1} \texttt{diag}_i(W) \circ \texttt{rot}_i(x),$$

where $\texttt{diag}_i$ is the i-th diagonal of the matrix represented as a vector, $\texttt{rot}_i$ is the ciphertext rotations by i slots to the left, and $\circ$ represents the Hadamard product of the vectors. Given a negative value i, the ciphertext is rotated to the right. For the BSGS optimization, we find the closest two factors $t = t_1 \cdot t_2$, and perform:

$$W \cdot x = \sum_{i=0}^{t_1-1} \texttt{rot}_{(i \cdot t_2)} \left(\sum_{j=0}^{t_2-1} \texttt{rot}_{(k)} \left(\texttt{diag}_{(k+j)}(W) \right) \circ \texttt{rot}_{(i)}(x) \right),$$

where $k = (-\lfloor (i \cdot t_2 + j)/t_2 \rfloor \cdot t_2)$. This optimization requires a total number of $t_1 + t_2 - 2$ rotations, t multiplications, and $t - 1$ additions in the encrypted domain. Thus, the closer the factors t_1, t_2 are, the higher the performance improvement.

Convolutional layer We adapt the 1-dimensional convolution implementation from CryptoTL [WSH+22]. They provide a one-dimensional CKKS implementation of the packed convolutional layer presented by Gazelle [JVC18]. This algorithm enables using a stride parameter to determine the level of granularity in filter applications and allows multiple filters to be applied sequentially. The output is flattened into the ciphertext vector by default when using multiple filters. In other words, a multi-dimensional tensor is mapped into a single vector within the ciphertext. This flattening requires one rotation per filter dimension. Given each filter with size f, we further require $f - 1$ rotations. These numbers match the two-dimensional case in Table 3 from Gazelle [JVC18]. We will discuss the specific numbers in Section 4.

Pooling Layer We apply average pooling layers as they are particularly "HE-friendly". Given a stride of $s = 1$, where the average is applied to all elements within the filter size f, we compute $\mathbf{x}_p = \frac{1}{f} \sum_{i=0}^{f} \texttt{rot}_{-i}(\mathbf{x})$. This procedure computes the average filter for all elements in the SIMD-encoded ciphertext. Typically pooling layers are used after convolutional layers. When applying multi-dimensional convolution filters, we already described that the output is a flattened matrix. Given a flattened matrix, we must not average elements of different channels. We accommodate our procedure and apply the factor $1/f$ to the elements of a channel and 0 to the remaining elements. Then we compute the average pooling for this channel only and shift it to the new position, as each channel shrinks by $f - 1$ elements, we need to shift the $j - th$ channel by $j \cdot (f - 1)$ to the left to uphold the flattened encoding. Ultimately, the required operations increase linearly in the number of channels. Thus, the pooling layer's complexity relies on the number of input elements and the number of channels.

Classification result The networks we employed end with the Softmax activation function. As this step does not contain trainable parameters, the client can execute it after decrypting the result. This setup saves us a costly approximation with minimal cost for the client side.

3.3 Encoding

As described in Section 2.1.1, we can encode $n/2$ elements into a single ciphertext where n is the polynomial degree of the ciphertext domain R_q. In the literature, we refer to this as the number of slots of the ciphertext. In the BSGS algorithm, we have intermediate values that take up twice the slots of the matrix dimension (recall that we use padded square matrices to compute the dense layers). Next to the dense layers, the convolutional layers also increase the used slots. However, while the dense layers use the additional slots only during computation, the convolutional layer increases the output dimension depending on the number of filters. Generally, looking at the maximum blowup of used slots after a convolutional layer or during a dense layer is essential. Let $\texttt{max} = m \cdot P$ be the maximum with the expansion factor m and the number of PoIs P. We can encode as many PoI into our ciphertext such that the maximum does not exceed the available slots. If we have more PoI per trace than that, we can encode the traces into multiple ciphertexts or increase n. However, the latter approach will also increase key sizes, encryption, and decryption runtimes.

Given the above recommendations on parameter selection [ACC+19, APS15, Bar16], we work with $n = 16\,384$ for most settings. Given the expansion factor, we can support $\frac{n}{2 \cdot m}$ PoI in one ciphertext. If the number of PoIs is much lower, we can execute multiple classifications simultaneously. Let us consider the maximum used slots $\texttt{max}$ in a given network and let $s = n/2$ be the total ciphertext slots. It is possible to execute $\left\lfloor \frac{s}{\texttt{max}} \right\rfloor$ classifications at once. Given a trace with 256 PoI, a factor $m = 8$, we could execute 4 trace classifications simultaneously.

3.4 Analysis Strategy

Following the generic introduction in Section 2.2, we now specify how we analyze our homomorphic predictions to reconstruct a key. We train our model to specific target labels in one-hot encoded form depending on the scenario. For an unmasked implementation, it is sufficient to set the label of an input trace to the Hamming weight of the S-box output $y_{i,j} = HW(\text{S-box}(p_j \oplus k_j))$, where i is the trace index and j is the attacked byte of the key. Given a masked implementation, we do not apply the Hamming weight as this would make our model dependent on the specific mask in the unknown mask scenario. Thus, our labels are the byte values of the S-box output $y_{i,j} = \text{S-box}(p_j \oplus k_j)$, and the classification results of the model are 256 probabilities, one for each potential S-box output value. We obtain a prediction matrix with 9 (in the unmasked scenario with the Hamming weight model) or 256 probabilities (in the masked scenario with the Identity model) for each attacked trace and compute the element-wise natural logarithm.[1] Then, we calculate the S-box output for each key hypothesis with the plaintext corresponding to the attacked trace. We create a hypothesis vector, storing the probability of the S-box output to the index of the corresponding key hypothesis. We sum up the log-likelihoods in this hypothesis vector for multiple attacked traces. Finally, we compute the rank, i.e., the index of the correct key in the hypothesis list sorted by its prediction probabilities. The key rank indicates the difficulty of recovering the key byte after classifying a specific number of traces. When the key rank converges to zero, an attacker can read of the key byte directly.

[1] The number of probabilities is decided based on the leakage model and the cryptographic function.

4 Experiments and Evaluation

Given the general description of our approach, we apply it to different models and datasets to evaluate the performance. Our experiments focus on two settings: first, we assess models designed explicitly with HE in mind, with power traces of an unprotected AES implementation obtained with the aid of a ChipWhisperer device[2]. Second, we apply our setup to a state-of-the-art approach using CNNs to attack protected AES implementations [ZBHV19]. The second experiment gives us insights into performance in real-world applications and the adaptation strategies to make neural networks "HE-friendly". We elaborate on "HE-friendliness" further in Section 2.1.1. Finally, we evaluate our experiments regarding runtime, memory usage, accuracy, and key recovery speed. The latter is the required number of traces to recover the attacked key byte reliably.

4.1 Experiment Setup

Environment We run our experiments on a system with AMD Ryzen 9 7900X 12-core Processor, 4.7GHz clock speed, 128 GB Memory running Ubuntu. Regarding implementation, we use TensorFlow v2.13 [Dev23] and scikit-learn v1.3 [PVG+11] for the plain deep learning Python implementations to train and test our networks. We use SEAL v4.1 [SEA23] for their CKKS homomorphic encryption implementation and CryptoTL [WSH+22] as a baseline for our encrypted layer implementation. We demonstrate our attacks using two different datasets containing the side-channel traces of AES-128 implementations.

Datasets The Chipwhisperer dataset gives a standard comparison base for the evaluation of different algorithms [OC14]. The dataset is obtained from running an unprotected AES-128 implementation on the Chipwhisperer CW308 Target. This dataset targets the first byte in the first round of the AES S-box with a fixed key. The dataset contains 3 000 traces, each consisting of up to 5 000 PoIs for the same key. As discussed in Section 3, the pre-processing phase selects the 20-2 500 best-fitting PoIs for the training dataset for each trace. Additionally, we also use the full dataset with 5 000 PoIs (without pre-processing) to train our classification model. When the training dataset consists of between 20 and 2 500 PoIs, the CKKS modulo degree is set to 16 384. However, due to the BSGS algorithm, when using 5 000 PoIs, the CKKS modulo degree is set to 32 768.

The ASCAD dataset is generated by taking measurements from an ATMega8515 running masked AES-128 and is proposed as a benchmark dataset for SCA [PSB+18]. The dataset consists of 50 000 profiling traces and 10 000 attack traces, each trace consisting of 700 features. In this paper, we use 45 000 profiling traces as training data and 5 000 as test data. The profiling and attacking sets both use the same fixed key.[3] We follow common naming convention for this dataset and denote it as `ASCADf`. We attack the third key byte as that is the first masked byte, and we consider the Identity (ID) leakage model. The dataset is provided on the ASCAD GitHub repository.[4]

Training When training a network for the ChipWhisperer dataset, we used an MLP and a CNN. After evaluating different hyperparameters for performance and "HE-friendliness", we display the selected network in Figure 1 with two dense layers and the approximated ReLU function described above. In our benchmarks, we tune the PoIs and the output size of the first fully connected layer. We depict our results in Table 1. Regarding training, we use the Adam optimizer and categorical cross entropy as the multi-class classification loss

[2] https://www.newae.com/chipwhisperer

[3] Current state-of-the-art results indicate that the ASCAD version with random keys is not significantly more complex than the fixed key version [PWP22].

[4] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_fixed_key.

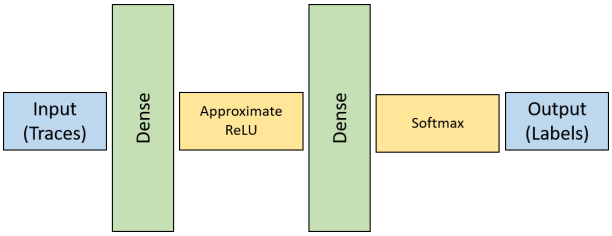

Figure 1: Multilayer perceptron architecture used in our ChipWhisperer experiments. This model is built with homomorphic classification in mind (HE-friendly).

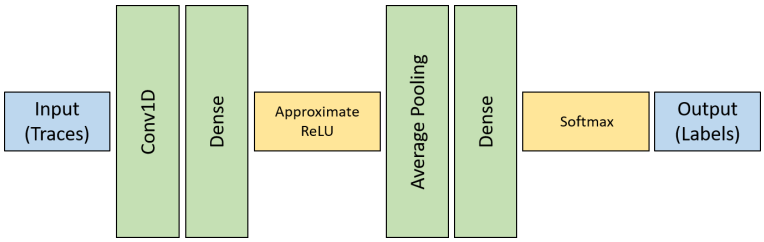

Figure 2: Shallow convolutional neural network designed for our ChipWhisperer experiments. In this model, layer parameters and activation functions are chosen with homomorphic classification in mind.

function. We train our MLP model with 300 epochs.

The CNN contains convolutional, dense, and average pooling layers. As activation functions, it builds on the approximate ReLU in the hidden layers and Softmax in the output layer. An overview is shown in Figure 2. The kernel size of the convolutional layer is set to 9, the average pooling layer has a pool size of 3, and the last dense layer has nine outputs for each output label. To remain "HE-friendly", we set the stride parameter in the convolutional and the average pooling layers to zero. Further, we use only one filter for the ChipWhisperer dataset in the convolutional layer. We use the Adam optimizer and categorical cross entropy as the multi-class classification loss function. We train our CNN model with 50 epochs.

As described above, we also follow the model architecture of [ZBHV19]. We have depicted the original structure in Figure 3. While their model is optimized for training and classification runtime, it naturally does not take "HE-friendliness" into account. After the ChipWhisperer dataset, where we designed a model directly, we now need to adapt an existing model to HE needs. Step by step, we replaced problematic parameters to evaluate the trade-off between accuracy loss and complexity in the encrypted domain.

Table 1: Training

Model	PoIs	Memory Usage	Time	Accuracy	Model Size	Weight Size	Dense Out
MLP	20	511 MB	16 s	0.5	16 KB	7.1 KB	20
	250	970 MB	11 s	0.8	34 KB	61 KB	20
	500	1.1 GB	10 s	0.8	53 KB	120 KB	20
	1000	1.1 GB	9 s	0.89	92 KB	240 KB	20
	2500	1.2 GB	9 s	1	502 KB	1.5 MB	50
	5000	1.4 GB	8 s	1	2 MB	6 MB	100
CCN	20	950 MB	15 s	0.3	23 KB	7.1 KB	–
	250	1 GB	15 s	0.9	274 KB	753 KB	–
	500	1.1 GB	19 s	1	1 MB	3 MB	–
	1000	1.2 GB	29 s	0.89	3.9 MB	12 MB	–
	2500	2 GB	172 s	0.89	24 MB	75 MB	–
	5000	3.8 GB	482 s	0.69	96 MB	302 MB	–

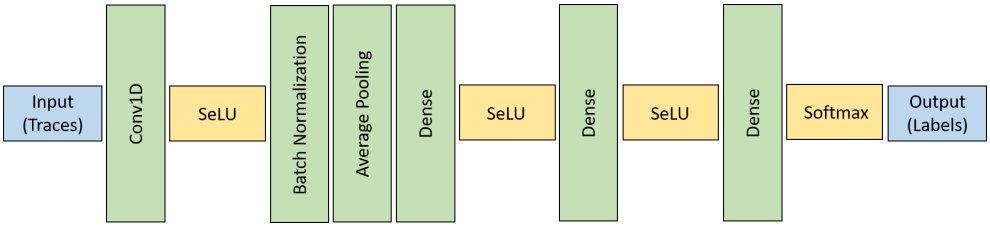

Figure 3: Optimized convolutional neural network used to attack the ASCAD dataset [ZBHV19]. We adapted the layers, activation functions, and layer parameters to accommodate homomorphic classification.

We applied all changes to the model architecture to the training and classification phase. We use the approximate ReLU function from Eq. (2) for the activation functions in the hidden layers. We removed the batch normalization layer, which boosts learning speed and prevents overfitting next to other aspects. Our experiments found that a dropout layer was sufficient to compensate for the overfitting control. Further, the stride parameter of the pooling layer led to some issues. One could calculate the average pooling as usual and then shift the relevant elements to their new position. However, this approach would take $c_{out} - 1$ homomorphic rotations, where c_{out} is the flattened output size of the average pooling layer. Ultimately, we set the stride parameter to one, as this did not impact the classification performance. In more complex layers, where the stride is more relevant, it is possible to implement the average pooling as a matrix-vector multiplication and enjoy the optimizations of the BSGS algorithm. In summary, all our experiments with the ASCAD dataset run with the network consisting of a convolutional layer, an average pooling layer, and three dense layers. The convolutional layer has four filters with a size of 1. The average pool has a stride of one and a kernel size of two. The three dense layers start with an input dimension of $d = 4 \cdot P - 4$, have ten neurons each, and output 256 key probabilities.

4.2 Evaluation

When training our model using MLP (Table 1), we witness a somewhat linear growth in the training time. Interestingly, we observe a decrease in the training time as we use more PoIs. We believe as the training data gets larger, more information is available to the network, leading to a faster saturation while training and thus stopping earlier before reaching the highest epoch. We do not witness similar behavior when training with CNN. For CNN training, we achieve a somewhat quadratic growth both in terms of memory and time with respect to the number of PoIs. However, in the CNN model section of Table 1, we can see that despite the exponential growth, our base memory consumption for a server is very low, especially if we consider that training the model is part of the one-time pre-processing.

For the CNN training, we conclude that the optimal number of PoIs for training the model is 500, as datasets with more PoIs lead to less accurate models due to unfit input, which otherwise gets filtered out during the pre-processing step. We also provide the model and the weight sizes that need to be stored on the server side and is used for private classification. Naturally, smaller models lead to more efficient homomorphic classification. The last column in Table 1 describes the output size of the first dense layer of the MLP network that is adjusted according to the increasing PoI sizes for better accuracy.

Table 2 shows the results for our MLP and CNN classification, respectively. The total private memory usage states the memory required by the server to perform a single client query containing 10 traces. For both MLP and CNN, we achieve a somewhat quadratic memory and time complexity regarding the number of PoIs that were used during the model training. Interestingly, for CNN classification, when using 5 000 PoIs, we observe a

Table 2: Classification

Model	PoIs	Accuracy	Memory Usage (Private SCA)	Avg. Query Time (Private SCA)	Avg. Query Time (Plain SCA)	Avg. Query Time Overhead
MLP	20	0.3	721 MB	310 ms	0.001 ms	$31 \cdot 10^4$
	250	0.9	1.2 GB	1.3 s	0.084 ms	$9 \cdot 10^4$
	500	1.0	2.7 GB	2.4 s	0.237 ms	$1.0 \cdot 10^4$
	1000	0.9	9.1 GB	5.1 s	1.4 ms	$0.4 \cdot 10^4$
	2500	0.9	20.1 GB	12 s	13 ms	$0.09 \cdot 10^4$
	5000	0.7	16.3 GB	39 s	72 ms	$0.05 \cdot 10^4$
CNN	20	0.5	210 MB	141 ms	0.001 ms	$14 \cdot 10^4$
	250	0.8	540 MB	681 ms	0.009 ms	$75 \cdot 10^4$
	500	0.8	790 MB	1 s	0.019 ms	$53 \cdot 10^4$
	1000	0.9	1.1 GB	2 s	0.04 ms	$50 \cdot 10^4$
	2500	1.0	2.4 GB	4.7 s	0.1 ms	$4.7 \cdot 10^4$
	5000	0.2	12.3 GB	25 s	0.2 ms	$12.5 \cdot 10^4$

drop in memory consumption. This is due to the matrix-vector product in the dense layer using the BSGS optimization as discussed in Section 3 leading to non-optimal matrix size when using 2 500 PoIs. Furthermore, we also compare the time difference between the plain and the private classification, giving us a better overview of the efficiency aspect of our private SCA when compared to a plain SCA. We observe that the private implementation is slower up to an order of magnitude of 4 in comparison to the plain evaluation of the network. However, we argue that since the base time requirement for the private SCA is reasonably low and practical for most servers, especially when using PoIs until 2 500, our design provides significantly better privacy benefits at a relatively minor cost. The last column of the classification table denotes the accuracy of predicting the correct Hamming weight when finding the key byte.

Notably, our privacy-preserving homomorphic SCA approach enables the classification of a key byte reliably with at most 10 traces and recovers the full key in *linear* time w.r.t. the security parameter of AES.

ASCAD evaluation After testing our implementation on the ChipWhisperer dataset, we focused on attacking the masked implementation. For comparability, we set up the optimized model [ZBHV19] to have a baseline for our implementation. Then we applied our adaptations to make the model "HE-friendly" as described in Sections 3.2 and 4.1. We evaluated our model in plain to have a baseline for the impact of our changes. Finally, we ran the encrypted classifications with 128-bit and 256-bit security. For more information on the security level or the classification procedure, we refer the reader to section 3.2. A comprehensive description of the key recovery process can be found in Section 3.4.

In Figure 4, we show the development of the key rank, averaged over 100 runs. Most notably, the private and plain classifications yielded reasonably similar results. This fact shows the feasibility of our approach in terms of accuracy. When comparing our model to the unadapted version, we see faster convergence. However, this difference is caused by a slight change in the setup. Usually, for each attack run, a different subset of 300 traces is selected from the 10 000 classifications. Due to the relatively high cost of HE classification, we fixed the 300 attack traces for all parties and only shuffled them with a pre-fixed seed. Although this approach ensures consistent results, the slight difference shown in the figure could be a random occurrence rather than a significant factor.

Nevertheless, we can see in Table 3 that the accuracy of our model is very close to the original one. The higher accuracy with 256-bit security is best understood with Table 4. This table shows the different CKKS parameters we used for our models. First, there are the two settings used in the ChipWhisperer attacks. Then, the settings that are used for our ASCAD model. The choice of $n = 32\,768$, as seen in the 4th row, gave us a little more budget for our q, so we could increase the integer precision γ. Recall that the maximum size of q depends on the security level and the polynomial degree n. It seems reasonable

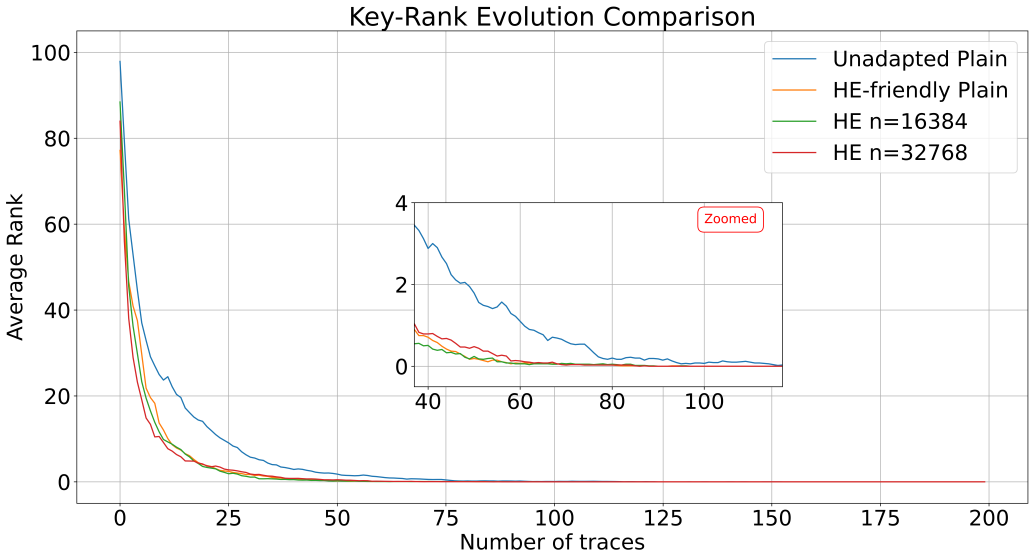

Figure 4: Comparison of key recovery attack on the ASCAD dataset with different approaches. Unadapted Plain uses the optimized model from [ZBHV19] and performs classification in plain. HE-friendly Plain use the adapted model but executes the classification in plain, while HE applies private classification with different polynomial degrees n and different security levels. All attacks use the same training data and classification data.

that doubling the security level leads to roughly twice the runtime and memory costs.

5 Discussion

5.1 Performance and Accuracy

The results of Section 4.2 demonstrate that private SCA and state-of-the-art DL-based SCA exhibit similar performance in terms of accuracy and convergence speed. As shown in Figure 4, HE does not limit the convergence speed. More precisely, given an "HE-friendly" neural network, we can select parameters that incur no accuracy penalty compared to plain evaluations. Of course, the adaptation to "HE-friendliness" might cost precision.

Contrary to accuracy, the primary distinction lies in the computational runtime needed to obtain these results. The private SCA requires a significantly longer computation time than the simple SCA, with a difference of four to five orders of magnitude, depending on the network architecture. It has been demonstrated in [BHM+20] that GPUs can be employed to expedite HE-related computations by one or two orders of magnitude, thereby

Table 3: Benchmark data for the ASCAD dataset. We test the state-of-the-art model against our adapted HE-friendly version. All runs evaluated the same attack traces. We ran the adapted model in plain and in HE with 128 and 256 bit security levels.

Model	Security Level	Accuracy	Memory Usage (Private SCA)	Avg. Query Time	Avg. Query Time Overhead
[ZBHV19]	None	0.013	-	0.03 ms	-
Our Model	None	0.010	-	0.03 ms	-
	128 bit	0.003	14GB	13.3 s	$4.5 \cdot 10^5$
	256 bit	0.010	27GB	27.4 s	$9.4 \cdot 10^5$

Table 4: CKKS parameters for our HE classifications. We see the results for both datasets we tested and both security levels evaluated. Next to the security level, we have the polynomial degree n, the bit size of the coefficient modulus q, the required multiplicative depth L, the fixed-point precision Δ and the integer precision γ

Dataset	Security Level	n	$\log_2 q$	L	Δ	γ
CW dataset	128 bit	16 384	360	6	40	20
	256 bit	32 768	360	6	40	20
ASCAD	128 bit	16 384	430	11	30	20
	256 bit	32 768	450	11	30	30

substantially mitigating the additional overhead. Similarly, there have been dedicated hardware optimizations for HE computation. In [MAK+23], the authors presented a successful port of CKKS to the FPGA platform. They achieve similar speedups with a single FPGA running at 2.3 GHz. Consequently, future research will prioritize the acceleration of computationally intensive operations. Next to hardware acceleration, we discussed potential speedups with parallelization. The deeper the network gets, the larger the CKKS parameters have to be. This property makes the execution of several queries in parallel more feasible the more complex the network gets.

HE can better adapt to wider networks than deep ones. We have seen that shallow networks allow for the classification of traces with 5 000 PoIs in 40 seconds. Exploring SCA applications that test these boundaries with private classifications is an open challenge. Considering the boundaries in neural network depth, we have seen several studies that either tackle this issue with colossal parameter settings [BHM+19] or the costly bootstrapping operation [LKL+22]. Another option is to integrate interactive procedures, where the client has an increased network cost to reduce complexity on the server side, as in [JVC18]. Given all these costly approaches, the first goal should always be the reduction of the network depth.

5.2 Trust Model

Homomorphic encryption paves the way for data protection during computation. However, discussing the trust assumptions more precisely is necessary to understand its security guarantees. Our system works in the semi-honest trust model. We assume parties that want to obtain input from the other parties but will never actively deviate from the protocol. In other words, given the correct use of the CKKS algorithms for key generation, encryption, decryption, and encrypted operations, input privacy is guaranteed up to the specified security parameter.

If one party deviates from the protocol, several potential problems arise. A malicious client can send random traces, so the final result will never identify any key leakage. Further, a client might send specifically crafted input traces to leak the private model parameters of the server [ACG+16]. Targeting the former issue requires verification of the measured traces, which is a challenging problem when combined with privacy. The privacy of the model can be boosted with differential privacy (DP) [DMNS06], a method to add fine-tuned noise to a dataset to limit the privacy loss caused by publishing (output-) data. Given a malicious server, a client cannot verify whether the correct model was executed. However, verifiable computations are a very active line of research [GGP10, FKL16, CKPH22, VKH23].

It is important to acknowledge that although these issues play a significant role in ensuring the practicability of HE-assisted DLSCA, they are outside the scope of this work. Consequently, addressing them would necessitate further research and analysis beyond the scope of this feasibility work.

5.3 Limitations and Conditions

Our work shows the general feasibility of HE-assisted DL-based SCA. Evaluating state-of-the-art models for side-channel leakage in the encrypted domain is possible. However, our approach requires a server with access to training data. Without this precondition, multiple problems arise. First, we require encrypted learning when preventing the server from learning the model. Then, the model is encrypted under the key of the data provider. In case just the training data but not the model is private, the model parameters can be decrypted. If the model should remain private as well, we need to employ multi-key HE [CCS19, CDKS19, MTBH21] settings that allow operations of ciphertexts encrypted by different parties. Such an approach would make it necessary for the data provider to play an active role in the classification part, which might be unrealistic depending on the setting. Finally, HE classification would get much more expensive if the model weights and biases were encrypted. Encrypted weight matrices cannot be used in the same BSGS-fashion. A matrix-vector multiplication with an $n \times n$ matrix in column-wise ciphertext encoding would need n ciphertext-ciphertext multiplications, $n \cdot \log_2 n$ rotations and $n \cdot \log_2 n$ ciphertext-ciphertext additions. However, different encoding strategies might improve performance.

6 Conclusions and Future Work

This work delved into the practicability question of employing homomorphic encryption in conjunction with neural networks to conduct private side-channel analysis. The main goal of this work was to identify the strengths and limitations of this approach with respect to the performance and accuracy of deep learning-based side-channel analysis results using HE-encrypted data. This study represents the first documented attempt to appraise the potential of AI-enabled HE-assisted side-channel analysis and points out the limitations of such an approach, including the need to trust the data sources. Our feasibility analysis using modern HE protocols and libraries suggests that already now, both unprotected and protected implementations can be attacked while traces remain encrypted. For further scaling towards larger use cases, we also identify high-level strategies to remain "HE-friendly": avoiding the increase in the depth of the neural network too much while allowing for "wider" neural networks is an approach that will work well with the current most suitable CKKS homomorphic encryption scheme.

In future work, it would be interesting to further consider the impact of hiding countermeasures (e.g., desynchronization) or noisy datasets (e.g., hardware implementations and/or software implementation with increased noise levels) on HE-encrypted data analysis to assess how well HE handles various noise sources. These problems are conventionally tackled with deeper neural networks. Such networks might be tackled by larger HE parameters or the client's involvement in the HE computation. It is possible to refresh intermediate CKKS ciphertexts by masking them and sending them to the client, who decrypts and re-encrypts before sending them back. With an interactive setup, it is also possible to directly evaluate non-linear functions with other secure multi-party computation approaches [JVC18]. Since the DLSCA community is not constrained by conditions on whether the neural network architecture is wider or deeper, the HE preference towards wider architectures is an interesting direction to investigate in the future.

Next to more complex models, the flexibility of our setting would be significantly improved by private learning. The server would then become agnostic to the hardware specifics as it cannot access the training *and* classification data. However, we already highlighted that next to computational overhead, it seems impossible to find a solution that does not require the data owners' participation during classification. Otherwise, a colluding client and server might leak the learned model. Thus, exploring options to limit

the data owners' involvement is relevant. Finally, it would be interesting to evaluate our setting where we use GPU/FPGA to speed up the computationally intensive operations for the specific case of side-channel analysis.

References

[ACC+19] Martin R. Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin E. Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption standard. *IACR Cryptol. ePrint Arch.*, page 939, 2019.

[ACG+16] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. pages 308–318, 2016.

[AEH15] Louis J. M. Aslett, Pedro M. Esperança, and Chris C. Holmes. Encrypted statistical machine learning: new privacy preserving methods. *CoRR*, abs/1508.06845, 2015.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

[Bag19] Karim Baghery. On the efficiency of privacy-preserving smart contract systems. In *AFRICACRYPT*, volume 11627 of *Lecture Notes in Computer Science*, pages 118–136. Springer, 2019.

[Bar16] Elaine Barker. Recommendation for key management, part 1: General, 2016-01-28 2016.

[BBH+22] Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. Privately connecting mobility to infectious diseases via applied cryptography. *Proc. Priv. Enhancing Technol.*, 2022(4):768–788, 2022.

[BCIV17] Joppe W. Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling. In Marc Joye and Abderrahmane Nitaj, editors, *Progress in Cryptology - AFRICACRYPT 2017*, pages 184–201, Cham, 2017. Springer International Publishing.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325. ACM, 2012.

[BHM+19] Ahmad Al Badawi, Luong Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. Privft: Private and fast text classification with homomorphic encryption. *CoRR*, abs/1908.06972, 2019.

[BHM+20] Ahmad Al Badawi, Louie Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. Privft: Private and fast text classification with homomorphic encryption. *IEEE Access*, 8:226544–226556, 2020.

[BLN14] Joppe W. Bos, Kristin E. Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *J. Biomed. Informatics*, 50:234–243, 2014.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[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.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomophic encryption from TFHE. Cryptology ePrint Archive, Report 2019/116, 2019. https://eprint.iacr.org/2019/116.

[CDKS19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. pages 395–412, 2019.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 45–68, Cham, 2017. Springer International Publishing.

[CGGI18] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption over the Torus, 2018. Report Number: 421.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. 33(1):34–91, January 2020.

[CGL+20] Edward J. Chou, Arun Gururajan, Kim Laine, Nitin Kumar Goel, Anna Bertiger, and Jack W. Stokes. Privacy-preserving phishing web page classification via fully homomorphic encryption. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2792–2796, 2020.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *SAC*, volume 11349 of *Lecture Notes in Computer Science*, pages 347–368. Springer, 2018.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *Cyber Security Cryptography and Machine Learning*, volume 12716 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2021.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT (1)*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

[CKPH22] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, and Jean-Pierre Hubaux. Verifiable encodings for secure homomorphic analytics, 2022.

[CRR03] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*, pages 13–28. Springer, 2003.

[Dev23] TensorFlow Developers. Tensorflow, July 2023.

[DKS+19] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madanlal Musuvathi. EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation. *CoRR*, abs/1912.11951, 2019.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. pages 265–284, 2006.

[DSC+19] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 142–156, New York, NY, USA, June 2019. Association for Computing Machinery.

[FKL16] Cédric Fournet, Chantal Keller, and Vincent Laporte. A certified compiler for verifiable computing. pages 268–280, 2016.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[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 *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. pages 465–482, 2010.

[HS14] Shai Halevi and Victor Shoup. Algorithms in helib. In *CRYPTO (1)*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2014.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for helib. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 641–670. Springer, 2015.

[HS18] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in helib. In *CRYPTO (1)*, volume 10991 of *Lecture Notes in Computer Science*, pages 93–120. Springer, 2018.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *LNCS*, pages 249–264. Springer, 2012.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. pages 1651–1669, 2018.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[KSS10] Michael Kasper, Werner Schindler, and Marc Stöttinger. A stochastic method for security evaluation of cryptographic FPGA implementations. In *FPT*, pages 146–153. IEEE, 2010.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[LKL+22] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.

[LKS17] Wenjie Lu, Shohei Kawasaki, and Jun Sakuma. Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. In *NDSS*. The Internet Society, 2017.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A Machine Learning Approach Against a Masked AES. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2013. Berlin, Germany.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. pages 1–23, 2010.

[LYS15] Wen-Jie Lu, Yoshiji Yamada, and Jun Sakuma. Privacy-preserving Genome-wide Association Studies on cloud environment using fully homomorphic encryption. *BMC Medical Informatics and Decision Making*, 15:S1, December 2015.

[LZC+21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 235–274, 2021.

[MAK+23] Ahmet Can Mert, Aikata, Sunmin Kwon, Youngsam Shin, Donghoon Yoo, Yongwoo Lee, and Sujoy Sinha Roy. Medha: Microcoded hardware accelerator for computing on encrypted data. 2023(1):463–500, 2023.

[MHS+19] Oliver Masters, Hamish Hunt, Enrico Steffinlongo, Jack L. H. Crawford, and Flávio Bergamaschi. Towards a homomorphic machine learning big data pipeline for the financial services sector. *IACR Cryptol. ePrint Arch.*, page 1113, 2019.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[MTBH21] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. 2021(4):291–311, October 2021.

[OC14] Colin O'Flynn and Zhizhang David Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2014.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[PCP20] Guilherme Perin, Lukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):337–364, Aug. 2020.

[PHG17] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. Template attack versus bayes classifier. *J. Cryptogr. Eng.*, 7(4):343–351, 2017.

[PHJB19] Stjepan Picek, Annelie Heuser, Alan Jovic, and Lejla Batina. A systematic evaluation of profiling through focused feature selection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2802–2815, 2019.

[PHPG21] Stjepan Picek, Annelie Heuser, Guilherme Perin, and Sylvain Guilley. Profiled side-channel analysis in the efficient attacker framework. In *Smart Card Research and Advanced Applications: 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11–12, 2021, Revised Selected Papers*, page 44–63, Berlin, Heidelberg, 2021. Springer-Verlag.

[PPM+23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Comput. Surv.*, 55(11), feb 2023.

[PSB+18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. Cryptology ePrint Archive, Report 2018/053, 2018. https://eprint.iacr.org/2018/053.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PWP22] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 828–861, 2022.

[RAD78] Ronald Rivest, Len Adleman, and Michael L. Dertouzos. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. pages 84–93, 2005.

[RO04] Christian Rechberger and Elisabeth Oswald. Practical template attacks. In *International Workshop on Information Security Applications*, pages 440–456. Springer, 2004.

[RWPP21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):677–707, Jul. 2021.

[SA21] Ravital Solomon and Ghada Almashaqbeh. smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 133, 2021.

[SBBV22] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin T. Vechev. Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In *IEEE Symposium on Security and Privacy*, pages 179–197. IEEE, 2022.

[SEA23] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.

[SKS09] François-Xavier Standaert, François Koeune, and Werner Schindler. How to compare profiled side-channel attacks? In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings*, volume 5536 of *Lecture Notes in Computer Science*, pages 485–498, 2009.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. pages 443–461, 2009.

[VJH21] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. Sok: Fully homomorphic encryption compilers. In *IEEE Symposium on Security and Privacy*, pages 1092–1108. IEEE, 2021.

[VKH23] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption, 2023.

[WJT+17] Shuang Wang, Xiaoqian Jiang, Haixu Tang, Xiaofeng Wang, Diyue Bu, Knox Carey, Stephanie Dyke, Dov Fox, Chao Jiang, Kristin Lauter, Bradley Malin, Heidi Sofia, Amalio Telenti, Lei Wang, Wenhao Wang, and Lucila Ohno-Machado. A community effort to protect genomic data sharing, collaboration and outsourcing. *SSRN Electronic Journal*, 01 2017.

[WPP22] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. *IEEE Transactions on Emerging Topics in Computing*, 2022.

[WSH+22] Roman Walch, Samuel Sousa, Lukas Helminger, Stefanie N. Lindstaedt, Christian Rechberger, and Andreas Trügler. Cryptotl: Private, efficient and secure transfer learning. *CoRR*, abs/2205.11935, 2022.

[WZD+16] Shuang Wang, Yuchen Zhang, Wenrui Dai, Kristin Lauter, Miran Kim, Yuzhe Tang, Hongkai Xiong, and Xiaoqian Jiang. HEALER: homomorphic computation of ExAct Logistic rEgRession for secure rare disease variants analysis in GWAS. *Bioinformatics*, 32(2):211–218, January 2016.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. 2020(1):1–36, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8391.