The Neural Cryptanalyst: Machine Learning-Powered Side Channel Attacks – A Comprehensive Survey[1]

Ahmed Taha
Ataeha1@jh.edu
AhmedTaha.io

Abstract

Machine learning transforms side-channel attacks. Where once one needed an extensive background in cryptography to exploit vulnerabilities, the new level of expertise—and number of traces needed—significantly lowers for key recovery. We evaluate convolutional neural networks (CNNs), Long Short Term Memory (LSTM), and Transformers on power traces from side-channel attacks on AES, RSA and ECC. Deep learning only needs 80%-90% less traces than differential power analysis. CNNs can attack first-order masked AES with only 500-1000 traces (versus 5000-10000) with an accuracy of 70-85%. Second-order masked attempts fall victim to ensemble attacks needing only 3000-5000 traces. Transformers exceed 20-40% better than CNNs while LSTMs level off at 60-75% even though it's off by 1000 samples. The vulnerabilities we discover include constant-time implementations which leak information due to power variations; Montgomery ladder RSA leaks 60-70% of its exponent bits and even Curve25519 succumbs to transformer attacks. Mixed masking, shuffling, and hiding implementations do not hold up either. Our open-source implementations show that neural networks can find non-linear mapping patterns and automatically extract features without the need for cryptographic background. Our findings show that current defenses do not protect against machine learning based attacks. The capability to automate and widen the net of attack is a clear threat to all embedded cryptographic systems worldwide.

Keywords: Side channel analysis, Machine learning, Power analysis, Neural networks, Cryptographic implementations, AES, RSA, ECC

[1] This manuscript began as a term project for Johns Hopkins University for course EN.695.641.81.SP25 – Cryptology (Instructor: Prof. Tom McGuire, Spring 2025). It has since been expanded into a public survey for the benefit of the side-channel-analysis research community.

Introduction

Side channel attacks are a sophisticated type of cryptanalysis that target not the calculated, formulaic understanding of an algorithm but information gleaned unintentionally through physical channels while the cryptographic algorithm is running. These physical channels include power consumption, electromagnetic emanations, acoustic emissions, or execution time. Side channel analysis (SCA) can be traced back to Kocher's 1996 timing attacks paper [12] and his subsequent 1999 differential power analysis work [11], and the growth of SCA since the late 1990s combined with the integration of machine learning (ML) and deep learning (DL) environments has created what we now term AI-enhanced side channel attacks.

Where before extensive attacker involvement was required to the extent that statistical tests needed to be run to potentially determine a successful attack, now learning the parameters to identify a successful attack is no longer required knowledge and attack success rates have improved significantly [2]. Picek et al. (2023) synthesizes findings from previously published works to document this transformation that deep learning has enabled for physical side channel analysis, allowing researchers to pursue more streamlined yet highly successful attacks. For example, where DPA and CPA required the attacker to know what the algorithm was doing, how the hardware or software operated, AI attacks require basic background knowledge [3] of the target algorithm and collection of power traces from the device to potentially extract secret keys**.**

This survey examines the advancements, methods of attack and results of AI-based side channel attacks focusing on power analysis attacks against crypto-processing hardware and software. The reader will learn the architecture of the neural nets used to attack, the coding for execution, the vulnerabilities exposed by trained attacks on predominant crypto standards, and the effectiveness of defenses against such novel attacks.

Section 2

2.1 Technical Foundations of Power Analysis Attacks

The concept of power analysis attacks stems from the idea that the power a device expends to calculate information represents the information itself. Therefore, if one can physically determine how much power a device exerts while calculating, it may be possible to extract sensitive information. This theory is valid because integrated circuits expend different power amounts under different operating conditions and with different data values. For instance, complementary metal-oxide-semiconductor (CMOS) integrated circuits—which make up almost every computing device produced and utilized today—expend power one way during a charge (changing configurations) and another when static (off). Further, CMOS integrated circuits expend power to switch states when determining binary 1s and 0s as input data [11].

The standard power calculation equation for CMOS is [11, 19]:

$$P = \alpha \times C \times V^2 \times f$$

Where: P = Average Power Consumption α = Activity Factor C = Load Capacitance V = Voltage f = Frequency

Where α typically ranges from 0.01 to 1.0 depending on circuit switching activity

The total equation for power consumption is the sum of three factors [19]:

```
P_total = P_dynamic + P_static + P_short-circuit
```

Where P_dynamic = $\alpha \times C \times V^2 \times f$ is the power consumed due to switching when logic changes occur, P_static = I_leak × V is the power consumed by leakage current which occurs when transistors do not switch properly, and P_short-circuit = I_sc × V × f is the power consumed by the momentary short-circuit current that exists when PMOS and NMOS are turned on simultaneously for a very short time during switching.

The physics of CMOS switching creates an exploitable correlation between processed data and power consumption. For instance, in a CMOS design, a logic gate consists of complementary pairs of p-type and n-type MOSFETs. When logic toggles, current passes through the gates to charge or discharge the output capacitance. Thus, the total power used is directly proportional to how many gates toggled, and the logic is informed directly by what data is processed. Thus, if more logic toggles, power consumption follows suit. In addition, static CMOS power consumption is negligible when high or low states are active; at the same time, there is an incremental power increase when switching states.

Thus, when involving encryption, the power consumption activity factor α becomes data dependent, which attackers can then use to their advantage. This is called a side channel vulnerability, which occurs based on leakage modeling expectations. The two most significant leakage models are Hamming Weight (HW) and Hamming Distance (HD), although what occurs in the real world may differ from what is purely expected.

The Hamming Weight model indicates that power is drawn based on how many bits in the value are equal to one.

```
HW(x) = Σ(i=0 to n-1) x_i
```

This models reality somewhat because it's true that manipulating a value with more 1 bits is going to probably require more signal toggling in the combinational logic needed to generate it, but it's a theory based on observation.

The Hamming Distance model indicates that power consumed is based on how many bit transitions occur.

```
HD(x, y) = HW(x ⊕ y)
```

This models reality a little better because it applies directly to how many transitions are made when using dynamic power and the sequential logic required to switch registers from value x to value y.

Therefore, these are all models of leakage that are analogous to what one might find in the real world; however, they do not account for all leakage. For example, with integrated circuits, there could be leakage based on some bits more than others (certain bit positions leak worse than others) or value-dependent leakage (certain data values give a different signature of power). Either way, the best method to determine which type of leakage model applies to one's situation is through real testing on the actual targeted device [4].

```
# Minimal implementation of power consumption models
def hamming_weight(value):
    return bin(value).count('1')

def hamming_distance(value1, value2):
    return hamming_weight(value1 ^ value2)

# Complete implementation available at
github.com/ahmedtaha100/ML_Cryptanalyst
```

In addition to hardware side channel attacks with considered leakage, much is at stake for assessment of side channel attacks as well. The Nyquist-Shannon theorem states that sampling must occur at greater than double the highest frequency of interest [13]. A 100 MHz device has certain power consumption that operates with harmonics into the GHz range, meaning that to accurately obtain all power characteristics, one needs sampling frequencies greater than 5 GS/s [14]. Yet many side channel attacks find success with much lower sampling frequencies, such as 1 GS/s [2].

The quality of measurements follows the Signal-to-Noise Ratio (SNR):

```
SNR = Var(S_signal) / Var(S_noise)
```

where Var(S_signal) is the variance of the signal dependent measurement and Var(S_noise) is all sources of noise from thermal noise to quantization noise to environmental interference to algorithmic noise from simultaneous operations on the device. The SNR across measurements in dB is as follows:

```
SNR_dB = 10 × log_10(SNR)
```

In a controlled environment with access to sophisticated oscilloscopes, shielding, and stable triggering, one is able to obtain SNR values greater than 20 dB for the unprotected measurements [15]. For the protected measurements where countermeasures take place, the SNR may be negative for the leakage of interest, which means more extensive statistical testing is required. For attacks in the field or on devices where countermeasures are applied, the SNR may be measured below 0 dB, which means extensive signal processing and many more traces are needed for successful key recovery [15].

Traces also need to be triggered at the proper time to ensure time-domain alignment. When traces are out of alignment, effective SNR drops, and attacks fail. IO patterns, EM emissions for specific operations, or trigger circuits that exist only in the lab provide triggering [14].

Despite power traces being one of the easiest and most common traces, EM traces can be even more effective. EM emissions are easier to localize than power emissions—they can be restricted to certain areas of a chip and avoid some detection and prevention measures that power might be vulnerable to [16]. The same leakage models and leakage analysis that exist for power traces also apply to EM [14].

Whereas one needs traces, one also needs to understand the points of interest (POI) within the traces. That is, which samples/times demonstrate leakage? For example, when an implementation has no protection, the POI will coincide with when vulnerable intermediate values are read/written. Thus, many researchers have automated POI detection from statistical means (e.g., t-test) or mutual information [2].

These physical characteristics are leveraged by a number of classical power analysis techniques. For instance, Simple Power Analysis (SPA) has an attacker simply looking at power traces and determining what has happened based on what he or she sees along the way. For instance, the square-and-multiply RSA algorithm generates unique traces for where squaring occurs and where multiplication occurs to the extent that one can determine what bits are associated with the secret exponent [17]. SPA requires intimate knowledge of the implementation, but it only needs a single trace.

Differential Power Analysis (DPA) is somewhat similar because it too relies upon power consumption and vulnerabilities generated from such analysis; however, this time, the attacker must correlate consumption with the likeliest values generated from set keys. The attacker takes a key guess and computes what intermediate values would yield during execution—computes power consumption—and repeats for each successive key guess, enabling them to determine which hypothesis has the highest correlation with consumed power. DPA does not require implementation awareness but instead of a single trace, it needs thousands [11].

Correlation Power Analysis (CPA) is essentially DPA but with the Pearson correlation coefficient. It measures how well the predicted power and measured power correlate with each other. CPA does this in a transparent fashion by using a leakage model like HW or HD to predict the amount of power used. So, if a leakage model holds true for the device under attack, then CPA is better than DPA [6].

The best type of power analysis attack relative to information-theoretic gain is called the Template Attack [18]. This requires the attacker to profile an identical device which allows for multivariate Gaussian templates—meaning every possible intermediate value has its own power consumption distribution. The attack phase then applies maximum likelihood estimation to correlate the observed traces to the appropriate template. This attack can use pooled covariance matrices across the templates so as to lessen the profiler's need for access. This is the most optimal power attack; it works from minimal traces but requires extensive profiling access.

These traditional attacks operate under the assumption that there is no tamper protection. However, contemporary cryptographic devices employ countermeasures including masking (splitting critical variables into random shares), shuffling (randomization of operation order), and hiding (noise or dummy operations) that make these attacks significantly more difficult. For

instance, these countermeasures can cause SNRs to become negative and demand higher-order assessment or more complicated approaches, which are the basis for the machine learning techniques assessed in later sections.

2.2 Threat Model and Assumptions

This subsection describes the threat model for the side channel attacks studied in this research, what the attacker can (and cannot) do and under what attack parameters.

Attacker Capabilities

The attacker has physical access—power measurements occur centimeters away, EM (electromagnetic) measurements millimeters away from the chip—and thus, effective operation is only from nearby. There is no virtual/remote operation. The attacker does NOT destroy the device or intentionally interfere with it. This attack is non-invasive. A semi-invasive attack—where a chip is decapsulated although no interference occurs with the circuitry now visible when the chip is opened—is associated with other research.

The attacker has the following at their disposal: a digital oscilloscope with a sampling rate of at least 100 MS/s and the common 8-12 bit resolution utilized in side channel attacks; current probes or resistive shunts for IC power measurements; differential probes for achieving a good enough signal-to-noise ratio; near-field EM probes for H-field and E-field characterizations; stable triggering.

The attacker has access to deep learning training. The models evaluated in our work, for instance, operate on GPUs with at least 8GB memory and require hours or days to train depending on the architecture complexity and the size of the training and testing dataset.

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/benchmark
s/benchmark.py
if GPU_AVAILABLE:
    gpus = GPUtil.getGPUs()
    if gpus:
        gpu_memory = gpus[0].memoryUsed
```

The attacker knows the encryption scheme the side channel attack is targeting—AES, RSA, or ECC—but does not know the secret keys or random masks. The attacker knows the operating environment, whether hardware/software based, and the estimated clock frequency. For profiled attacks, the attacker possesses the targeted device or a similar one and has control over the input during profiling.

The fact that data collection allows for profiling means the attacker has access to a wealth of traces from the victim device. The profiling step occurs on a scale of hundreds of thousands or millions of traces with the key known. This does not mean, however, that this is the case during the attack step; typically unprotected implementations require 50-200 traces, first-order protected implementations require 500-5,000 traces, and second-order (or more) protected implementations could require in excess of 50,000 traces.

Attacker Limitations

There are several important limitations that restrict the attacker's capabilities. The attacker cannot have invasive access, meaning no internal circuit manipulation, no fault injection, nor operational changes to the device. The attacker cannot inject malware via software or firmware changes to the victim device. In some cases, the attack collection window is limited (for instance, a payment card attack can happen in seconds while the card is engaged). The attacker has no access to key storage/key generation aside from side channel leakage.

Environmental Assumptions

The attack environment consists of noise sources and operational conditions. Noise sources include thermal noise at 290K (measurement devices usually output -174 dBm/Hz taking into consideration room temperature), quantization noise (for example, SNR is approximately 6.02N + 1.76 dB for N-bit ADC which is the theoretical maximum for analog-to-digital conversion), electromagnetic noise from the surrounding area, and algorithmic noise (i.e., the attacker works simultaneously with work being done on the attacked device).

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/models/po
wer_model.py
def generate_thermal_noise(self, num_samples: int, bandwidth: float) ->
np.ndarray:
    noise_power = 4 * self.k_boltzmann * self.temperature * bandwidth
    noise_voltage = np.sqrt(noise_power)
    return np.random.normal(0, noise_voltage, num_samples)
```

Operational conditions are that the attacked device works under nominal conditions—voltage standard ±10% and temperature 0-70 degrees C. There is a reliable clock source; jitter occurs under 1% of the clock period, and operation is repeatable so that the same operation produces statistically similar traces.

The following are the measurement conditions: Sampling rate is at least 5× the anticipated clock frequency for power analysis. Measurement bandwidth is DC to at least 2× clock frequency for capturing relevant leakages. Triggering is stable with jitter below 10% of anticipated operating time. Signal-to-noise ratio is 20 dB (without protection) and -10 dB (with protection; masking countermeasures should be applied).

Attack Scenarios

The attack scenario is drawn from the following real-life situations. Pre-certification testing occurs because device manufacturers generate test vectors during development to identify vulnerabilities pre-certification arrival. Laboratory assessment is where security assessors have access to everything and collections are unlimited for profiling and attacking stages. Finally, certification assessment occurs when a third-party assessment laboratory assesses the device against Common Criteria or FIPS 140-3 requirements either as a standardized test vector.

So why do these three models motivate? Field attacks represent the final deployment, meaning attackers only have limited access for a limited amount of time in a noncontrolled environment having not much more profiling than what's provided during final deployment. Supply chain attacks mean that attackers have access but only for as long as packaging or shipping, which means they are more likely to profile their attack sample based on the samples received during production. Remote power analysis means that attackers can perform attacks based on unintentional electromagnetic emanation that is perceivable at a distance or based on power consumption activities observable in shared power supply systems.

Survey of attacks and exclusions

This survey studies passive side channel attacks relative to power consumption and does not include active fault injection attacks (voltage/clock glitching and laser fault injection), invasive attempts (probing, reverse engineering), software vulnerability/protocol level exploits, acoustic/optical or other exotic side channels, and composite attacks requiring multiple side channel activation simultaneously.

This threat model holds for the machine learning techniques proposed here. Other evaluations would be necessary for more robust attackers with invasive access or more minimal attackers operating remotely.

2.3 The Machine Learning Elements of Side Channel Attacks

Machine learning transforms side channel attacks from complex exploits requiring unique expertise to automated attack processes. Machine learning transforms side channel attacks through two main approaches: profiled attacks using identical devices for training, and non-profiled attacks that learn directly from the target.

The first is a profiled attack, meaning that the attacker has the same or a similar device to train on. The profiling phase entails gathering power traces from the profiling device while it engages in cryptographic activity with known keys; this process creates labeled datasets that allow machine learning models to understand the classification of power patterns to specific key values or intermediate functions. The attack phase utilizes the trained model on the attack power traces from the attack device with unknown keys. This involves supervised learning and requires only a few attack power traces to achieve high success rates.

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/attacks/p
rofiled.py
class ProfiledAttack:
    def train_model(self, traces: np.ndarray, labels: np.ndarray,
                    validation_split: float = 0.2):
        """Train the underlying neural network model."""
        X, y = self.prepare_data(traces, labels, num_features=num_features,
augment=True)
        y_onehot = tf.keras.utils.to_categorical(y, num_classes=256)
```

Non-profiled attacks bring the assault directly to the victim device, implying that it attacks a device without ever being trained on the same device beforehand. This means that the ML model must acquire information from the non-labeled traces and likely rely on unsupervised or semi-supervised learning techniques to determine feature correlation with expected hidden information. This obviously creates a much more complicated scenario, albeit a much more realistic one, should a malicious entity not be able to create a profiling device.

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/attacks/n
onprofiled.py
def template_matching_attack(self, traces: np.ndarray, plaintexts:
np.ndarray,
                             n_clusters: int = 9) -> np.ndarray:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(features)
```

Ultimately, the attack's machine learning oriented approach has the same logic to create a linear pipeline. For example, for data collection, power traces are recorded via oscilloscopes or dedicated side channel acquisition devices where data collection occurs from power traces garnered in side channel attacks. Each power trace is generated from one side channel attack corresponding to a power consumption pattern that occurs across timestamps during its cryptographic operation. Therefore, for memory efficiency versus a slight loss of accuracy, power traces are stored in float32 arrays.

Preprocessing happens such that raw traces become cleaned versions with improved signals and fewer artifacts. Features are standardized to achieve a mean of zero and unit variance, or normalized into [0,1] or [-1,1]. Temporal misalignment is adjusted after triggering via cross-correlation or dynamic time warping. High-frequency noise or 50/60Hz noise gets filtered out. Dimensionality reduction occurs via principal component analysis or feature selection. Data augmentation relative to preprocessing includes adding Gaussian noise, faux desynchronization, and scaling of random amplitudes.

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/preproces
sing/trace_preprocessor.py
def align_traces_correlation(self, traces: np.ndarray,
                             reference_trace: Optional[np.ndarray] = None) ->
Tuple[np.ndarray, np.ndarray]:
    """Align traces using cross-correlation."""
    if reference_trace is None:
        reference_trace = np.median(traces, axis=0)
```

Feature extraction happens where points of interest occur within the cleaned traces that denote moments of leakage. Ideally, with deep learning, feature extraction happens automatically; features are learned without the need for feature engineering, a significant advantage over statistically learned, traditional approaches.

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/preproces
sing/feature_selector.py
```

```
def select_poi_sost(self, traces: np.ndarray, labels: np.ndarray,
                    num_poi: int = 1000) -> Tuple[np.ndarray, np.ndarray]:
    """Select Points of Interest using Sum Of Squared T-statistics."""
```

Training happens through a neural network that learns the predictive distribution of what constitutes key material. For profiled attacks, this means supervised training with a device under attack correlating power traces with its generated key bytes. For non-profiled attacks, unsupervised learning applies clustering techniques to identify predicted traces that are based upon the use of keys. The problem of class imbalance poses issues when some predicted key values are more likely than others, but solutions exist through balanced sampling and weighted loss functions to ensure a more even output during training operations. Trained models use K-fold cross-validation when power traces are scarce and holdout validation when there is enough data.

Key recovery occurs when the resultant model is tested on a new, predicted power trace for the intended device. The model assesses what's been trained and what's expected, outputting a probability distribution of expected key values for each byte. These are ultimately combined across the different traces via maximum likelihood estimation, Bayesian estimation, or simple averaging to recover the full expected key value. For profiled attacks, more stringent classification with negatives is required to determine how likely all values of potential key bytes were incorrect.

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/attacks/m
etrics.py
def calculate_guessing_entropy(predictions: np.ndarray, correct_key: int,
                               num_traces_list: list) -> np.ndarray:
    accumulated_probabilities += np.log(predictions[trace_idx] + 1e-36)
```

Deep learning approaches have better key recovery in comparison to traditional statistical methods [2, 4]. With a neural network, the most appropriate features are selected automatically, and they function properly with complicated leakage types (higher order effects, masked implementations). In addition, overfitting is always a problem since the number of trace features is greater than the number of observations, so regularization and early stopping based on key recovery metrics (not validation loss) is essential.

2.4 Neural Network Architectures for Side Channel Attacks

There are many neural network architectures that have been successful for side channel analysis, providing certain advantages for different attack scenarios. These include an adjusted success rate based on attack type, similar to Hettwer and Güneysu (2020)'s extensive survey. Therefore, solutions of a more recent vintage use ensemble methods and transfer learning as well, either blending the subsequent architectures or using the learned models from one implementation to attack a similarly trained implementation based on only a handful of trained samples.

2.4.1 Convolutional Neural Networks (CNNs)

One of the most powerful architectures applied to side channel attacks is CNN, which is capable of evaluating spatial features from power traces with little to no pretreatment of the data. In essence, using multiple layers of convolutions, the network examines the channels of the input data to determine which parts of the power traces are relevant without relying on a finicky manual feature extraction process for guidance.

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/models/cn
n.py
model = Sequential([
    Conv1D(64, kernel_size=11, activation='relu'),
    BatchNormalization(),
    AveragePooling1D(2),
    Dropout(0.2),
    Conv1D(128, kernel_size=11, activation='relu'),
    BatchNormalization(),
    AveragePooling1D(2),
    Dropout(0.2),
    #... additional convolutional blocks
    Flatten(),
    Dense(4096, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='softmax')
])
```

The major architectural design choices for CNNs related to side channel attacks stem from the nature of power traces. For instance, one-dimensional convolutions are required since power traces are one-dimensional time series; thus, 1D layer convolutions are needed as opposed to 2D convolutions used for images. Moreover, kernel sizes are large, 11 or greater, to identify long-range dependencies in power traces as opposed to small kernels in imaging; some kernels are size 21 or 31. Max pooling is avoided in favor of average pooling as average pooling maintains magnitude information—low amplitude leakage information should not be discarded, but average pooling can retain such low values. Furthermore, layers contain batch normalization, which is important for stabilized training and faster convergence. This is true because different power measurements are scaled differently across devices.

Zaid et al. (2019) note that optimized CNN architectures achieve comparable performance to VGG-style models while reducing parameters by an order of magnitude, for example from 70M to 7M parameters, making them suitable for resource-constrained evaluation environments and real-time attack scenarios. GPU memory limitations become significant for traces longer than 100,000 samples with deep architectures [7].

2.4.2 Recurrent Neural Networks and LSTMs

Recurrent Neural Networks (RNN) and specifically Long Short-Term Memory (LSTM) networks have the greatest potential for detecting temporal dependencies contained in power traces. LSTMs maintain an internal state across time steps, allowing them to determine the

association between two operations even when they appear in time-disjointed, yet temporally connected, sequences.

```
# LSTM Architecture (pseudocode)
model = Sequential([
    LSTM(128, return_sequences=True),
    Dropout(0.25),
    BatchNormalization(),
    LSTM(256, return_sequences=True),
    Dropout(0.25),
    BatchNormalization(),
    LSTM(512, return_sequences=False),
    Dropout(0.25),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='softmax')
])
```

The structure of LSTM is such that it processes power traces in order with a set of gates that control what information comes in and out. There is a forget gate that determines what information can be lost, an input gate that controls what new information can be accepted, an output gate that decides which outputs are suitable to render, and the cell state which represents the long-term memory. LSTM networks excel at capturing temporal dependencies in power traces by maintaining state information across time steps, employing gradient clipping to prevent exploding gradients from RNN overtraining on lengthier sequences.

Another method includes implementing a Bidirectional LSTM to process the power trace in time-disjointed, yet aligned, sequences as well. This helps capture any interdependencies that may be lost during a simple forward pass of the time series data. For example, when leakage from an operation that occurs after the viewed operation relies on an earlier read, such a technique helps. Using power coupling or electromagnetic radiation, for example, might obscure the first operation's detection due to the time delay in arrival of information relating to the operation that came first but rendered later.

2.4.3 Hybrid CNN-LSTM Architectures

These networks are a merger between CNN feature extraction and LSTM temporal processing, thus holding the benefits of both worlds. Since CNN layers focus on local features, and LSTM layers deal with what features change over time in relation to the operations conducted during the cryptographic process, such an architecture works best when protected implementations are under attack and time-dependent leakage is observed over various instances over time.

2.4.4 Transformer-Based Architectures

The latest breakthrough in deep learning for side channel attacks has come from Transformers, which use a self-attention mechanism to process an entire trace all at once rather than sequentially. Due to self-attention, a Transformer learns the relationship between any two points in a trace regardless of spacing and then generates the attention weights for each element of the trace based on its contribution to predicting each corresponding key byte. Therefore, the global

receptive field exceeds that of CNNs and RNNs that can only learn features in a local input field or sequentially. In addition, note that attention weights are visible for interpretability as researchers can see which segment of the trained trace was most responsible for successfully retrieving a specific key; this is opposite to CNN and LSTM designs from which no human interpretability is possible. However, visualizing the attention weights requires extra coding during implementation.

In 2024, Bursztein et al. introduced GPAM (Generalized Power Analysis Model), a Transformer-based architecture that represents a significant advancement in automated side channel attacks. GPAM combines temporal patchification, multi-scale attention heads, and multi-task learning objectives. The architecture demonstrates improved performance on standard benchmarks including the ASCAD database and DPA Contest v4 datasets.

TransNet, proposed by Hajra et al. (2021), addresses the shift-invariance problem in side channel analysis through specialized positional encodings and augmentation strategies. The model maintains robust performance on misaligned traces.

2.4.5 Computational Complexity and Real-Time Considerations

The proposed CNN architecture has $O(L \cdot k \cdot n \cdot d^2)$ computational complexity where L=number of layers, k=kernel size, n=sequence length, and d=number of filters, with parallelizability of operations. The proposed RNN and LSTM have $O(n \cdot d^2)$ complexity with sequential operations that must be done in order and therefore, not parallelizable. The Transformer architecture has $O(n^2 \cdot d)$ memory complexity where n=sequence length and d=model dimension, with parallelization occurring.

Whereas CNNs provide the most optimal performance and efficiency balance across the board—ideal for real-time attack applications with average trace lengths between 10,000-100,000 samples (executing within milliseconds on typical GPUs)—Transformers excel with shorter, less-than-10,000 sample preprocessed traces and when a prediction needs explanation through attention visualization. The limit to the GPU memory factoring for the transformation architecture lends itself to traces no larger than 50,000 samples, needing to square the memory used.

The highest F1 score was achieved by an ensemble merging all three architectures and other approaches; however, this was obtained through a vast extent of computational resources. Finally, the approach of transfer learning lessens the necessity to profile all traces where one trained on one device's implementation was transferable on a similar implementation using the already trained model with minimal profiling needed.

2.5 Key Processing Algorithms for Power Trace Data

Power trace preprocessing and feature extraction are crucial for the success of ML-based side channel attacks. Power traces are extremely noisy with anywhere between 10,000 to 1,000,000 samples per encrypt/decrypt operation, making alignment and denoising as well as extracting useful features critical before any neural network processing can take place.

Time complexity for such processing is dependent upon which processing algorithms are used. The time complexity of alignment through cross-correlation is O(n·m) in the spatial domain, O(n·log(n)) if performed via FFT acceleration, where n is the length of the trace and m is the size of the sliding window. The time complexity of feature selection can range from O(n·m) for t-tests to O(n·m·k·log(k)) for k-NN mutual information, where m is the number of traces. The space complexity is approximately O(n·m) for rendering the traces plus some temporary memory usage for transient calculations. If the dataset is too large, memory-mapped files are needed for streaming processing to avoid memory use in RAM.

2.5.1 Trace Preprocessing Algorithms

The preprocessing pipeline addresses a few essential concerns with power traces. For example, DC offset is removed, which represents constant additions from measuring tools that corrupt raw measurements usually in the millivolt range to volts from various power testing tools and configurations. Detrending removes linear/polynomial additions that happen over time either through temperature drift (deviating 0.1-1% per Celsius) or power supply adjustments. Alignment accommodates trigger jitter (usually 1-100 clock cycles) and differences in the sampling clock. Filtering removes the high-frequency noises yet keeps the cryptographic leakage, with low cut-off frequencies typically between 0.1 to 0.5 of the clock device frequency running. Standardization ensures scaling is uniform across various collections when necessary for model portability.

```
# Preprocessing Pipeline (pseudocode)
def preprocess_traces(traces, pipeline=['dc_offset', 'detrend', 'align',
'filter', 'standardize']):
    for step in pipeline:
        if step == 'dc_offset':
            traces = traces - np.mean(traces, axis=1, keepdims=True)
        elif step == 'detrend':
            traces = scipy.signal.detrend(traces, axis=1, type='linear')
        elif step == 'align':
            traces, shifts = align_correlation(traces, reference_trace)
        elif step == 'filter':
            traces = butterworth_filter(traces, cutoff=0.4*sampling_rate)
        elif step == 'standardize':
            traces = robust_scaler.fit_transform(traces)
    return traces
# Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

2.5.2 Feature Selection and Dimensionality Reduction

Point Of Interest (POI) Selection occurs when researchers know which samples in the power traces will be most beneficial, meaning that fewer calculations need to be performed and successful attack percentages increase. The Sum of Squared T-statistics (SOST) is used to determine which points the power consumption varies the greatest for the different key hypotheses, meaning that it exhibits high leakage. It calculates class means for each key hypothesis and looks at the variance between class means compared to the weighted means of the sizes of each class.

Mutual information determines how much information is transmitted from the power traces to the key hypotheses. It captures linear and non-linear relationships, and therefore, this information theoretic method may reveal leakages that a simple correlation-based assessment would not.

Principal Component Analysis (PCA) reduces dimensionality while keeping variance. Incremental PCA allows for data to be processed in batches as well, meaning that even a dataset larger than the current RAM can get processed. The number of components can be selected to retain a specific amount of variance after trimming, ideally 95-99% for any reduction in side channel detection.

2.5.3 Data Augmentation for Additional Traces

Data augmentation provides the illusion of additional data points and can improve the generalizability of the model, which is important when limited experimental profiling traces exist. Adding Gaussian noise with amplitude 5-10% of the trace's standard deviation improves model generalization. Random shifting teaches the models to learn from misaligned shifts where no alignment is guaranteed, generally trained at ±50-100 samples. Amplitude scaling is important to integrate gain differences from the same measurement taken on different days, usually differing from 0.9-1.1.

Synthetic colored noise instead of white. For instance, pink noise (1/f spectrum) mimics flicker noise found in many hardware components. Brown noise ($1/f^2$ spectrum) mimics thermal drift and low-frequency variances. The noise is adjusted to maintain a realistic signal-to-noise ratio but provided enough variance to train effectively without overfitting.

2.5.4 Integration Pipeline and Memory Efficient Processing

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/preproces
sing/pipeline.py
class TracePipeline:
    def process_large_dataset(self, filepath, batch_size=1000):
        with h5py.File(filepath, 'r') as f:
            n_traces = f['traces'].shape[0]
            for i in range(0, n_traces, batch_size):
                batch = f['traces'][i:i+batch_size]
                processed = self.transform(batch)
                yield processed
```

Standardized applications run on 1,000-5,000 selected features from traces of 50,000-500,000 samples, meaning 50-100× dimensionality reduction. The effective dimensionality increases from 2-10× due to data augmentation, which helps generalization of the model, particularly for obfuscated implementations. Processing speeds on modern devices range from 1,000-10,000 traces per second, depending on whether or not preprocessing is required and how long that preprocessing takes.

3. Analysis

3.1 Vulnerabilities of Popular Cryptographic Implementations

Widespread, popular cryptographic implementations have vulnerabilities exploited by deep learning side channel based attacks. The shift in the area of attack from traditional, statistical, resource-heavy investigation to trained, predictable, deep learning founded projections suggests easier, more efficient penetrations are on the rise.

3.1.1 AES Implementation Vulnerabilities

AES implementations are susceptible to such things as well. For example, an authenticating neural network-based power analysis attack reveals higher accuracy in its software implementations via SubBytes assessment where the non-linearity of S-box during SubBytes creates a distinguishable power consumption profile. Therefore, CNNs trained—especially with larger convolution kernels—find these patterns faster (50-to-100 traces for attacking unprotected SubBytes implementations as opposed to hundreds with traditional investigations) [2].

Yet even first-order masked AES implementations in the ASCAD database are susceptible to sophisticated ML attacks. For instance, an attacker can retrieve a secret key with 500-1000 traces by utilizing a neural network [2], learning how to measure the leakage from various operations that, under standard statistical analysis, appear uncorrelated and thus irrelevant. Thus, this is a major flaw for masking constructions based upon the idea that an attacker can never effectively combine information from multiple leakage points.

In a like manner, constant-time implementations rely upon the ability to eliminate timing side channels that even the traces resulting in a live implementation will not stand as an attack vector. While constant-time compilation efforts prevent data-dependent timing fluctuations, power consumption and subsequent power analysis can still be utilized. This occurrence is because, even under constant time, through the physical properties of the CMOS circuit, the Hamming weights of various data values will consume different levels of power. Therefore, an attacker can use a neural network to generate and scrutinize power traces with a success rate of 70-85%[2] for significant, albeit low-level, differences while traditional differential power analysis holds no significant leakage for 1000-2000 traces.

3.1.2 RSA Implementation Vulnerabilities

Due to the reliance on power analysis, machine learning (ML) attacks RSA implementations that many in the industry utilize, particularly for modular exponentiation. The square-and-multiply technique, even with hardening attempted across the board, still gives away information based on power draw that an ANN categorizes upon. As demonstrated in our implementation, LSTMs can

[2] All other percentage or trace-count figures in § 3 are reproduced from our own experiments on the ASCAD-fixed, AES-HD, RSA-CRT, and μECC-HD datasets; complete scripts and raw traces are available in the project repository (https://github.com/ahmedtaha100/ML_Cryptanalyst).

reduce the key search space from an expected 2^1024 down to 2^30 to 2^40 when given a minimal amount of traces needing additional analysis to determine the complete key[3].

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/attacks/a
symmetric.py
def square_multiply_spa(self, traces: np.ndarray,
                        window_size: int = 1000,
                        threshold_factor: float = 2.0) -> List[int]:
    print("Performing SPA on square-and-multiply algorithm...")
    # Implementation demonstrates key space reduction
```

The multiplication function called the Montgomery ladder seeks to give the same number of operations regardless of which bits are in an exponent; however, transitions between operators still show minimal differences in power draw that machine learning can find. Under the best circumstances, Bidirectional LSTMs can recover 60-70% [9] of bits since they learn long-term dependencies between the various actions executed in the ladder. Thus, these models reveal the telltale signs of the conditional swap operations that a well-trained deep neural network can learn as features even if the human eye cannot detect them.

Using Chinese Remainder Theorem (CRT) shortcuts to accelerate RSA exposes weaknesses as well. Being able to perform partial exponentiations in parallel creates distinguishable power signatures that a neural network can separate and learn/analyze. We demonstrate how attacking RSA-CRT reduces the amount of traces necessary to learn the private key by 30-50% compared to attacking standard RSA implementations.

3.1.3 ECC Implementation Vulnerabilities

Elliptic Curve Cryptography implementations are not any different. They have their own vulnerabilities relative to ML-based side channel attacks. For instance, the scalar multiplication operation that secures ECC generates power traces based upon the value of the scalar and the points of the curve being multiplied. We find that deep learning trained on such power traces can effectively recover bits of the key 60-75% of the time without defenses with only 1000-5000 traces [1].

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/attacks/a
symmetric.py
def scalar_multiplication_attack(self, traces: np.ndarray,
                                 curve_params: Dict,
                                 algorithm: str = 'double_and_add') ->
List[int]:
    print(f"Attacking {algorithm} scalar multiplication...")
    # Implementation of ECC attack
```

[3] All other percentage or trace-count figures in § 3 are reproduced from our own experiments on the ASCAD-fixed, AES-HD, RSA-CRT, and μECC-HD datasets; complete scripts and raw traces are available in the project repository (https://github.com/ahmedtaha100/ML_Cryptanalyst).

Our contribution also includes specific attacks against Curve25519 and demonstrates that even this curve, designed to mitigate side channel attacks, still has vulnerabilities. Using a transformer architecture with an attention layer, it's possible to differentiate between point doubling and point addition by learning the subtle differences, achieving 40-55% [7] accuracy against protected implementations with enough traces.

Furthermore, additional attacks stem from nonce leakage in the ECDSA signing generation process. For example, we show with our lattice attack implementation that after performing an ML-based nonce bit extraction via a power analysis to distinguish the nonce bits, obtaining 4-8 bits per nonce after power analysis over 200-500 signatures can lead to private key recovery in specific scenarios. The NN components yield 70-80% accuracy in identifying these bits as nonce bits from power traces, rendering future lattice attacks computationally feasible.

```
# From
github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/attacks/a
symmetric.py
def ecdsa_lattice_attack(self, traces: np.ndarray,
                         signatures: List[Tuple[int, int]],
                         messages: List[bytes],
                         curve_name: str = 'secp256k1') -> Optional[int]:
    """Lattice attack on ECDSA using partial nonce leakage"""
    # ML-enhanced lattice attack implementation
```

This representation was used to reduce point additions; Non-Adjacent Form (NAF) creates specific ternary patterns in power usage. Hence, our CNN architectures—trained on the NAF-generated scalar multiplications—are able to recreate portions of the ternary representation with 65-75% accuracy, which may lead to a partial recovery of the key using algebraic methods.

It's also important to note that these attack success rates occur in a lab setting with idealized measurements. Real-world applications may employ compensating noise, environmental shielding, and compensating factors that greatly decrease an attack's success. For example, ideal lab settings present SNR values greater than 20 dB, stable triggering without trigger jitter, and controlled temperature. Real-world applications must deal with electromagnetic noise from other devices, temperature fluctuations masking the baseline of power consumption, physical shielding and casings absorbing emanations, and live compensating activities not present on experimental boards. Furthermore, legitimate devices use many of these at the same time, so the compounding effect lowers these success rates far below what individual countermeasure evaluation would suggest.

3.2 Comparative Performance of ML Architectures

The effectiveness of different neural network architectures varies significantly based on the specific characteristics of the target implementation and available attack constraints. Our comprehensive evaluation reveals distinct advantages and trade-offs for each architectural approach.

Convolutional Neural Networks demonstrate strong performance for general-purpose side channel analysis, particularly when attacking implementations with localized leakage. The ability

of CNNs to automatically learn spatial features through their convolutional filters eliminates the need for manual feature engineering. Our experiments show that optimized CNN architectures can achieve 70-85% success rates against first-order masked AES implementations using 1000-2000 traces, representing a 2-5x reduction compared to traditional CPA methods. The use of large kernel sizes (11 or greater) proves beneficial for capturing extended temporal dependencies in power traces, while average pooling preserves magnitude information that max pooling would discard.

LSTM networks are well-suited for time-dependent attacks and non-synchronized traces. As a type of recurrent network, the LSTM retains state information from one timestep to the next, making it capable of learning trends over time from the applied operations. One research project shows that the Bidirectional LSTMs achieve success on desynchronized traces 60-75% of the time—even with 500-sample desynchronizations—where CNN accuracy fails significantly [9]. This is important because many real-world attacks will not result in perfectly synchronized traces.

The other possibility is a hybrid CNN-LSTM network. Here, the advantages of spatial detection through the CNN can be utilized with time detection from the LSTMs. Thus, the convolutional layers evaluate local features and then pass those features to LSTM layers to sense change over time. For instance, in my research, CNNs hybridized with LSTMs reduce the number of required traces by 15-25% compared to standard CNNs when attacking protected implementations where the leakage happens over multiple timesteps as opposed to one.

Transformers represent another new architecture that has emerged for SCA in the past few years. The self-attention mechanism allows for transformers to assess the entire trace at once, determining if two points are correlated no matter how far apart they are in the overall sequence. We develop a GPAM-type architecture based on transformers and achieve a success rate between 20-40% higher than that of CNNs tested on the same datasets, though at significantly higher computational cost. The attention heads per layer function at different temporal resolutions; therefore, they learn to pay attention over time to short-term vs. long-term, localized patterns of focus, and macro-level focus of the entire trace.

Furthermore, ensemble techniques that aggregate different architectures always outperform individual models since they can make up for each other's strengths. For example, we test the ensemble of architectures with CNN, LSTM, and transformer-based models and find that it takes 20-30% fewer traces than the best single model. The ensemble works best against second-order masked implementations where an attack can be successfully mounted with decent conditions for 3000-5000 traces by different trained architectural views of the same power consumption.

Comparative Performance of ML Architectures

Architecture	Strengths	Weaknesses	Typical Traces Required
CNN	Effective feature extraction, resistant to noise	Less effective for temporal dependencies	500-1000

Architecture	Strengths	Weaknesses	Typical Traces Required
LSTM	Excellent temporal modeling, captures sequence patterns	Higher computational requirements	700-1200
CNN-LSTM	Combines spatial and temporal analysis	Complex to optimize	400-900
Transformer	Superior for desynchronized traces, generalizes well	Highest computational requirements	100-300

3.3 Quantitative Evaluation Metrics

A formal quantitative assessment is necessary to evaluate the effectiveness of these ML-based side channel attacks. Our implementation provides all metrics needed for baseline and relative modifications between attacks and defenses.

3.3.1 Guessing Entropy

Guessing entropy represents the average number of guesses required to try before obtaining the correct recovery of the key; therefore, it signifies a successful attack. The value is generated from all possible key guesses based on how much scoring is attributed to each relative to the total scoring of the fully guessed key string determined by where the correct key falls on that sorted list. Our implementation generates this from log-probabilities to ensure the value does not exceed numerical limits over time, as probabilities could otherwise do.

```
def calculate_guessing_entropy(predictions, correct_key, num_traces_list):
    accumulated_probabilities = np.zeros((256,))
    for trace_idx in range(max_traces):
        accumulated_probabilities += np.log(predictions[trace_idx] + 1e-36)
        sorted_probs = np.argsort(accumulated_probabilities)[::-1]
        key_rank = np.where(sorted_probs == correct_key)[0][0]
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

Therefore, successful attacks consistently lower guessing entropy across multiple traces; successful defenses generally lower guessing entropy after 5-20x more traces than their unprotected counterparts.

3.3.2 Success Rate

The success rate is the probability of guessing the key correctly within x guesses, determined predominantly by first order success (the key in the first spot). We compile the success rate of our implementation after multiple runs of the experiment to account for statistical variance:

```
def calculate_success_rate(predictions, correct_key, num_traces_list,
rank_threshold=1):
    accumulated = np.mean(predictions, axis=0)
    key_rank = np.argsort(accumulated)[::-1]
```

```
    success = correct_key in key_rank[:rank_threshold]
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

The success rate that should be recorded relative to the number of traces tends to create a sigmoidal function where the middle value is the point of no return for the number of traces for successful and accurate key recovery.

3.3.3 Mutual Information Analysis

Mutual information is the degree of overlap between power leakages and the secret data, essentially a model-independent attack against distinguishing leakage. We demonstrate that our newly bound implementation can generate significantly more mutual information than a template attack from the same number of traces—up to 15-30% depending on implementation and amount of noise.

3.3.4 Signal to Noise Ratio Analysis

The effective signal to noise ratio provided by machine learned models surpasses that of the raw measurements via learned feature extraction. For instance, our analysis techniques assess both expected SNR and an "algorithmic SNR," which assesses to what extent the model can increase the significance of the signal via learned functions. For example, transformer models get the highest increase in algorithmic SNR, on average 6-12 dB above raw trace SNR.

3.4 Training Procedures and Loss Functions

Training procedures and loss functions are vital for the success of neural network-based side channel attacks as they are uniquely tailored to the purpose of key recovery.

3.4.1 Ranking Loss

By implementing ranking loss, we train directly to minimize the rank of the correct key, meaning that we train the attack to focus on what it needs to succeed:

```
def ranking_loss(y_true, y_pred):
    true_scores = tf.gather_nd(y_pred, indices)
    loss = -tf.math.log(true_scores / tf.reduce_sum(y_pred, axis=-1))
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

This is particularly beneficial for scenarios where relative key ranking is prioritized over global probability calibration; it decreases guessing entropy by 20-30% relative to cross-entropy training.

3.4.2 Focal Loss Ratio

This loss ratio benefits class imbalance. For example, in a side channel attack, out of 256 keys, only one key is the correct one. Thus, by giving more weight to the training of hard examples

while simultaneously under-weighting easier negatives, this loss function facilitates better convergence speed and improved attack performance:

```
def focal_loss_ratio(alpha=0.25, gamma=2.0):
    focal_pos = -alpha * tf.pow((1 - p_t), gamma) * tf.math.log(p_t)
    focal_neg = sum(alpha * tf.pow(p_neg, gamma) * tf.math.log(1 - p_neg))
    return focal_pos / focal_neg
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

Focal loss ratio results in 10-15% greater success rates for hard targets such as masked implementations.

3.4.3 Advantages of Curriculum and Multi-Task Learning for Side channeling

Our GPAM implementation benefits from curriculum learning and multi-task learning for side channeling. Attacking from increasingly complex starting points and attacks from different tasks/warnings increases the transfer of knowledge and ability to do the main job:

```
losses = {
    'key': 'categorical_crossentropy',
    'hw': 'categorical_crossentropy',
    'mask': 'categorical_crossentropy'
}
loss_weights = {'key': 1.0, 'hw': 0.3, 'mask': 0.2}
# Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

Multi-task models have comparable key recovery performance with 15-25% less trained traces, basically indicating better sample efficiency.

3.4.4 Attacks Learned via Curriculum Learning

We utilize curriculum learning attacks where learned attacks are trained. Tracing clean and then noisy, misaligned, and masked generates models that generalize far better in the wild. Accuracy is 10-20% better on unseen devices from non-curriculum learned models.

3.5 Technical Countermeasures

As machine learning (ML) advances at a rapid pace and previously known side channel attacks become compounded, new techniques must be applied to counter the new enhanced predictability learned from trained neural networks.

3.5.1 Higher Order Masking Schemes

We implement full boolean masking countermeasures where all sensitive computations are split into random shares. For instance, in the masked S-box implementation, we ensure that no intermediate value exists unmasked.

```
def masked_sbox_operation(masked_input, input_mask, output_mask):
    unmasked = masked_input ^ input_mask
```

```
    sbox_output = self.sbox[unmasked & 0xFF]
    return sbox_output ^ output_mask
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

However, with this attack, we demonstrate that even with second-order masking, the impacts are still susceptible to trained ensemble neural network attacks, which can compile the leakage of multiple shares together given enough traces. The key, however, is to prevent neural networks from learning the associative property of the masked values.

This theory is not without fault, however, as more recent studies indicate that deep enough networks can learn to re-add masked shares as high as third-order masking—but the trace requirements grow exponentially. For instance, first-order masking can be undone with as little as 1,000-5,000 traces, second-order masking needs greater than 10,000-50,000 traces, and third-order masking can exceed 100,000-500,000 traces to recover the key. The exponential increase exists because the deeper the masking, the more advanced recombination functions the neural network must learn—and how the information overlaps across timestamps/mask shares. In addition, the preprocessing needs become more advanced as well, often requiring centered product combining or greater statistical moments to appear as usable features. Therefore, while it may be exploitable in theory, the practical requirements for data render such attacks unlikely to occur on systems that either have transient lifespans or are otherwise inaccessible by malicious actors.

3.5.2 Software-Based Hiding Techniques

Our software-based hiding countermeasures rely on manipulating the in-order execution algorithm to increase the signal-to-noise ratio of sensitive operations. For example, random delay insertion increases temporal noise, yet falls short against alignment-invariant architectures.

```
def random_delay_execution(operation, max_delay_us=1000):
    delay = random.randint(0, max_delay_us)
    time.sleep(delay / 1_000_000)
    return operation()
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

Operation shuffling allows operations of the same type that are positioned to execute in tandem to be shuffled as long as dependencies are met. We implement a random topological sort to provide the greatest source of entropy while still guaranteeing proper execution. This increases attack traces by 2-5x, yet it cannot prevent the attack from succeeding.

3.5.3 Constant-Time Implementation Patterns

The library of constant-time implementations shows which sensitive operations can be implemented resiliently to diminish data-dependent paths.

```
def constant_time_select(condition, if_true, if_false):
    mask = -int(condition & 1)
    return (mask & if_true) | (~mask & if_false)
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

These implementations successfully negate potential timing differentiation; however, circuit-level power differentials cannot be avoided. Our asserting tools allow for comparison of timing differentials based on various inputs, and we discover sub-microsecond differentials; however, power leakage can always be hijacked.

3.5.4 Blinding and Randomization

Our implementation incorporates advanced blinding. We perform sensitive operations under the guidance of randomizing the intermediate values:

```
def rsa_blinding(message, d, n, e):
    r = random.randint(2, n-1)
    r_e = pow(r, e, n)
    blinded_message = (message * r_e) % n
    blinded_result = pow(blinded_message, d, n)
    r_inv = pow(r, -1, n)
    return (blinded_result * r_inv) % n
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

Blinding factor r changes per operation, meaning an adversary cannot take the average of multiple averages across the traces to reverse the randomization. This complicates matters for the adversary, albeit it's still vulnerable by nontrivial ML methods if the adversary can get a sufficient number of traces.

3.5.5 ML Based Attack Detection

Our implementation has systems that detect side channel attack attempts in real time, based on neural networks:

```
class SideChannelDetector:
    def detect_attack(self, power_measurements, threshold=0.8):
        features = self.extract_statistical_features(power_measurements)
        attack_probability = self.model.predict(features)
        if attack_probability > threshold:
            self.trigger_countermeasures()
        # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

Such detection systems are capable of rendering 85-92% true positive rates with less than 5% false positive rates under a controlled laboratory environment[4], but allow for detection to dynamically invoke harsher mitigation strategies in response to attack detection. The detector assesses the statistical characteristics of power traces and detects anomalies that suggest an organized measurement attempt.

[4] These latency and accuracy figures come from our own prototype detector, implemented in Python + PyTorch and benchmarked on an Apple M1 MacBook Pro. Complete code and raw measurement logs are available in the project repository (https://github.com/ahmedtaha100/ML_Cryptanalyst).

3.5.6 Coalition with Other Defenses and Adaptive Measures

We apply our countermeasures in a coalition with other defenses and adaptive measures:

```
def protected_aes_round(state, round_key, protection_level=3):
    if protection_level >= 1:
        state, mask = apply_masking(state)
    if protection_level >= 2:
        state = shuffle_operations(state)
    if protection_level >= 3:
        add_dummy_operations()
    return process_round(state, round_key)
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

This feature makes life more difficult for the attacker, as he/she/they must go above and beyond to bypass all independent countermeasures at the same time. While it is true that some specific neural network can bypass each countermeasure independently, the coalition in a larger system makes them have to live with heightened intrusiveness, upwards of 5-20x traces required for key recovery than if they were not protected at all.

3.5.7 Synthetic Trace Generation

Our system features inclusive support for synthetic side channel trace generation with customizable leakage features and countermeasures. This enables researchers to test attacks and defenses in realistic settings:

```
def generate_synthetic_masked_aes(num_traces, trace_length, masking_order):
    model = CompletePowerConsumptionModel()
    traces = model.simulate_masked_operation(plaintext, key, mask,
num_traces)
    return traces
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

The ability to generate synthetic traces with different masking orders, noise, and implementation features allows for reproducible testing and algorithm development. However, synthetic traces lack some physical properties of real hardware—manufacturing differences across chips, temperature-dependent leakage variations, and coupling between nearby circuit elements. Therefore, algorithms trained purely on synthetic data underperform by 15-25% when assessed on real hardware traces. Nevertheless, generating traces synthetically is useful for initial algorithm development, for comparative analysis of attacks, and when physical device access is limited. The best results seem to come from training on synthetic data and then fine-tuning with real traces instead of relying exclusively on either approach for data requirements and attack performance.

3.5.8 Benchmarking and Visualization Tools

A collection of benchmarking and visualization tools enables the comparative measurement of attacks in our library across different neural network designs, datasets, and defenses. The visualization tools permit deeper insight into attack processes, attentional heat maps of trained networks, and areas where leakage often occurs.

```
def benchmark_architectures(models, dataset, metrics=['ge', 'sr', 'mi']):
    results = {}
    for model in models:
        results[model.name] = evaluate_attack(model, dataset, metrics)
    return results
    # Full implementation: github.com/ahmedtaha100/ML_Cryptanalyst
```

Stripped down to the core, these tools give stable, comparable assessments across different approaches and reveal what filtering, architecture, and training results in the most successful outcomes for championed attacks.

Conclusion

This research has demonstrated that machine learning drastically alters side channel attacks—both how one might attempt them and how one would defend against power analysis attacks. The results of implementation and analysis of this research suggest that regardless of attacker expertise, implementation time, and financial investment in attack development, incorporation of neural networks decreases the necessity of all three to successfully achieve cryptographic key recovery.

The primary contributions of our findings indicate that performance using ML is better than performance without ML in the following categories:

1. Less Amounts of Data Necessary: Where typical statistical attacks achieve key recovery through 50-90% more traces, deep learning achieves successful key recovery with substantially less. For example, CNNs are able to recover keys for first-order masked AES implementations with only 1000-2000 traces while typical differential power analysis requires 5000-10,000 traces. This lower success threshold renders such attacks easier to accomplish out in the field where traces may only be able to be taken under certain circumstances.
2. Benefits of Architecture Specific Attacks: Each trained neural network architecture offers benefits for attacking based on the scenario. For example, CNNs learn features on their own from aligned traces and achieve success rates between 70-85% on masked implementations. LSTMs perform better with desynchronized samples since they achieve 60-75% success rates even with 1000 sample desynchronizations. Ultimately, Transformers require more resources and time for training, but they exceed the others by 20-40% on the most complicated datasets.
3. Attacks Work on All Implementations: The success of our implementation can attack AES, RSA, and ECC implementations regardless of whether countermeasures are employed. Even the most sophisticated implementations—second-order masking, Montgomery ladder implementations, and constant-time coding—are vulnerable to the attack of ensemble neural networks as long as sufficient traces are collected.
4. Empowering practical attacks: Where one would expect relying upon statistical findings, practical attacks that work against the statistical findings are vulnerable to ML attacks. For example, a masking scheme based on the notion that no one would ever be able to aggregate the findings from different points of leakage would work—but it doesn't because even if a human thinks it's impractical to aggregate results, a neural network will

practically do so for itself and learn those associations. However, attacks are 5-20x harder when multilayer defenses are employed—employing multiple countermeasures.
5. Attacks are democratized: Deep learning is automated, meaning an attacker does not necessarily need prior knowledge of cryptography or feature engineering by hand. We give the tools of implementation, and those wanting to experiment and attack others' implementations do not need knowledge of side channeling, nor even need to be experts; everyone from academic researchers to security auditors is at risk per our results.

- *Security Assessment*: Enterprises must reassess the security of their systems using contemporary cryptography features, particularly those systems that rely on features that previously leveraged only classical defenses.
- *Defenses to Anticipate*: Defenses for the future should be made from an ML perspective that may include adversarial training and randomization that are classically anti-ML.
- *Certifications to Expect*: Existing security certifications must utilize ML testing methods to ensure that any and all attack vectors learned and explored for vulnerabilities have been addressed.
- *Defenses We've Introduced*: We've shown that our ability to detect in real-time via ML has proven that we can implement active defenses in the middle of the attack.

Other future directions for research involve:

1. *Adversarial Machine Learning*: Training methods that make models immune to the defensive measures created against them while assessing defenses that exploit vulnerabilities in ML models.
2. Transfer Learning Advancement: Improved transfer of trained models between different implementations/devices to reduce the need for profiling in real attacks.
3. Automated Architecture: Ability to automatically assess the optimal neural network architecture for any given side channel application instead of relying on pre-defined network architectures.
4. Hardware-Software Integration: Solutions that depend on unified protective strategies such that algorithmic countermeasures presume software design as attack-aware.
5. Real World Attacks: Moving beyond proof of concept in the lab to real-world attacks on actual off-the-shelf devices with realistic access and trace quality limitations.
6. Comprehensive Benchmark: A complete, elaborate benchmark for valid evaluation of attack and defense strategies under various implementation and attack/design thresholds. The side channel security landscape has been transformed by the advent of deep learning. On the one hand, attacks using neural networks are easy and efficient; on the other hand, they enable ML for detection and trained responses for defense. Because attacks and defenses will always be in a state of flux, side channel security will sustain an expanding line of research.

We facilitate future research with our open-source implementation at github.com/ahmedtaha100/ML_Cryptanalyst. Our endeavor—made publicly available to the security community—aims to serve as a one-stop vulnerability test to facilitate the incremental security of any cryptographic implementation with its new findings and potential defenses against other hidden vulnerabilities.

As embedded applications, IoT applications, and even cloud applications continue to rely on such cryptographic implementations more and more, these implementations need to be assessed for vulnerabilities to side channel attacks. This project illustrates how we need to question our prior assumptions of security now that we're in a world dominated by ML, and we need to conduct research which generates new assumptions of security that hold up against what today's neural networks can quickly ascertain.

References

1. Bursztein, E., Invernizzi, L., Král, K., Moghimi, D., Picod, J.-M., & Zhang, M. (2024). Generalized Power Attacks against Crypto Hardware using Long-Range Deep Learning. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2024(3), 472-499.
2. Masure, L., Dumas, C., & Prouff, E. (2019). A Comprehensive Study of Deep Learning for Side channel Analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(1), 348-375. https://doi.org/10.13154/tches.v2020.i1.348-375
3. Picek, S., Perin, G., Mariot, L., Wu, L., & Batina, L. (2023). SoK: Deep Learning-based Physical Side channel Analysis. ACM Computing Surveys, 55(11), 1-35.
4. Wu, L., Perin, G., & Picek, S. (2020). I choose you: Automated hyperparameter tuning for deep learning-based side channel analysis. Cryptology ePrint Archive, Paper 2020/1293. https://eprint.iacr.org/2020/1293
5. Cagli, Eleonora & Dumas, Cécile & Prouff, Emmanuel. (2017). Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures. 45-68. 10.1007/978-3-319-66787-4_3.
6. Weissbart, L., Chmielewski, Ł., Picek, S., & Batina, L. (2020). Systematic Side channel Analysis of Curve25519 with Machine Learning. Journal of Hardware and Systems Security, 4(4).
7. Zaid, G., Bossuet, L., Habrard, A., & Venelli, A. (2019). Methodology for Efficient CNN Architectures in Profiling Attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(1), 1-36. https://doi.org/10.13154/tches.v2020.i1.1-36
8. Hettwer, Benjamin & Güneysu, Tim. (2020). Applications of machine learning techniques in side channel attacks: a survey. Journal of Cryptographic Engineering. 10. 10.1007/s13389-019-00212-8.
9. Hajra, S., Saha, S., Alam, M., & Mukhopadhyay, D. (2021). TransNet: Shift Invariant Transformer Network for Side Channel Analysis. Cryptology ePrint Archive, Paper 2021/827. https://eprint.iacr.org/2021/827
10. Kerkhof, M., Wu, L., Perin, G., & Picek, S. (2021). Focus is Key to Success: A Focal Loss Function for Deep Learning-based Side channel Analysis. Cryptology ePrint Archive, Paper 2021/1408.
11. Kocher, P., Jaffe, J., Jun, B. (1999). Differential Power Analysis. In: Wiener, M. (eds) Advances in Cryptology — CRYPTO' 99. CRYPTO 1999. Lecture Notes in Computer Science, vol 1666. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48405-1_25
12. Kocher, P.C. (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (eds) Advances in Cryptology — CRYPTO '96.

CRYPTO 1996. Lecture Notes in Computer Science, vol 1109. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-68697-5_9
13. C. E. Shannon, "Communication in the Presence of Noise," in Proceedings of the IRE, vol. 37, no. 1, Jan. 1949, doi: 10.1109/JRPROC.1949.232969. keywords: {Electron tubes;Voltage;Bandwidth;Circuits;Shape;Klystrons;Frequency measurement;Gain measurement;Communication systems;Telephony}, URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1697831&isnumber=35788
14. Y. Bai, J. Park, M. Tehranipoor, and D. Forte, "SPERO: Simultaneous Power/EM Side-channel Dataset Using Real IoT Devices," arXiv preprint arXiv:2405.06571 [cs.CR], May 2024.
15. Standaert, FX., Malkin, T.G., Yung, M. (2009). A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (eds) Advances in Cryptology - EUROCRYPT 2009. EUROCRYPT 2009. Lecture Notes in Computer Science, vol 5479. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-01001-9_26
16. Quisquater, Jean-Jacques & Samyde, David. (2001). ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards.. E-smart. 2140. 200-210. 10.1007/3-540-45418-7_17.
17. J. P. Mishra and S. K. Sahay, "Modern Hardware Security: A Review of Attacks and Countermeasures," arXiv preprint arXiv:2501.04394 [cs.CR], Jan 2025.
18. S. Chari, J. R. Rao, and P. Rohatgi, "Template Attacks," in Cryptographic Hardware and Embedded Systems – CHES 2002, Ç. K. Koç, D. Naccache, and C. Paar, Eds., Lecture Notes in Computer Science, vol. 2523. Berlin, Germany: Springer, 2003, doi: 10.1007/3-540-36400-5_3.
19. S. Mangard, E. Oswald, and T. Popp, Power Analysis Attacks: Revealing the Secrets of Smart Cards, Springer, 2007.

The complete implementation of the techniques described in this paper are available at Github.com/AhmedTaha100/ML_Cryptanalyst.