AI Report

We predict this text is

Human Generated

AI Probability

5%

This number is the probability that the document is AI generated, not a percentage of AI text in the document.

Plagiarism

?

The plagiarism scan was not run for this document. Go to gptzero.me to check for plagiarism.

The Neural Cryptanalyst: Machine Learning-Powered Side Channel Attacks A Comprehensive Survey - 6/19/2025

Ahmed Taha

The Neural Cryptanalyst: Machine Learning-Powered Side Channel Attacks - A Comprehensive Survey

Ahmed TahaAtaeha1@jh.eduAhmedTaha.io

Abstract

Machine learning transforms side-channel attacks.
Where once one needed an extensive background in cryptography to exploit vulnerabilities, the new level of expertiseand number of traces neededsignificantly 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

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 circuitswhich make up almost every computing device produced and utilized todayexpend 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]:

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

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

Where a 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]:

$$P_{total} = P_{dynamic} + P_{static} + P_{short\text{-}circuit}$$

Where $P_{dynamic} = a \times C \times V^2 \times f$ is the power consumed due to switching when logic changes occur, $P_{static} = I_{leak} \times V$ is the power consumed by leakage current which occurs when transistors do not switch properly, and $P_{short\text{-}circuit} = I_{sc} \times V \times f$ 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 a 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.

$$HW(x) = \sum_{i=0}^{n-1} x_i$$

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.

$HD(x, y) = HW(x \oplus y)$

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 x 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 emissionsthey 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 executioncomputes power consumptionand 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/benchmarks/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/power_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 conditionsvoltage 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 5x the anticipated clock frequency for power analysis.
Measurement bandwidth is DC to at least 2x 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/profiled.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/nonprofiled.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/preprocessing/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/preprocessing/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/metrics.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/cnn.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 informations 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 boardideal 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/preprocessing/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 50-100x dimensionality reduction.

The effective dimensionality increases from 2-10x 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 trainedespecially with larger convolution kernelsfind 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% 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 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.

```
# From github.com/ahmedtaha100/neural_cryptanalyst/src/neural_cryptanalyst/attacks/asymmetric.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/asymmetric.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
```

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/asymmetric.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 thro

FAQs

What is GPTZero?

GPTZero is the leading AI detector for checking whether a document was written by a large language model such as ChatGPT. GPTZero detects AI on sentence, paragraph, and document level. Our model was trained on a large, diverse corpus of human-written and AI-generated text, with a focus on English prose. To date, GPTZero has served over 2.5 million users around the world, and works with over 100 organizations in education, hiring, publishing, legal, and more.

When should I use GPTZero?

Our users have seen the use of AI-generated text proliferate into education, certification, hiring and recruitment, social writing platforms, disinformation, and beyond. We've created GPTZero as a tool to highlight the possible use of AI in writing text. In particular, we focus on classifying AI use in prose. Overall, our classifier is intended to be used to flag situations in which a conversation can be started (for example, between educators and students) to drive further inquiry and spread awareness of the risks of using AI in written work.

Does GPTZero only detect ChatGPT outputs?

No, GPTZero works robustly across a range of AI language models, including but not limited to ChatGPT, GPT-4, GPT-3, GPT-2, LLaMA, and AI services based on those models.

What are the limitations of the classifier?

The nature of AI-generated content is changing constantly. As such, these results should not be used to punish students. We recommend educators to use our behind-the-scene Writing Reports as part of a holistic assessment of student work. There always exist edge cases with both instances where AI is classified as human, and human is classified as AI. Instead, we recommend educators take approaches that give students the opportunity to demonstrate their understanding in a controlled environment and craft assignments that cannot be solved with AI. Our classifier is not trained to identify AI-generated text after it has been heavily modified after generation (although we estimate this is a minority of the uses for AI-generation at the moment). Currently, our classifier can sometimes flag other machine-generated or highly procedural text as AI-generated, and as such, should be used on more descriptive portions of text.

I'm an educator who has found AI-generated text by my students. What do I do?

Firstly, at GPTZero, we don't believe that any AI detector is perfect. There always exist edge cases with both instances where AI is classified as human, and human is classified as AI. Nonetheless, we recommend that educators can do the following when they get a positive detection: Ask students to demonstrate their understanding in a controlled environment, whether that is through an in-person assessment, or through an editor that can track their edit history (for instance, using our Writing Reports through Google Docs). Check out our list of several recommendations on types of assignments that are difficult to solve with AI.

Ask the student if they can produce artifacts of their writing process, whether it is drafts, revision histories, or brainstorming notes. For example, if the editor they used to write the text has an edit history (such as Google Docs), and it was typed out with several edits over a reasonable period of time, it is likely the student work is authentic. You can use GPTZero's Writing Reports to replay the student's writing process, and view signals that indicate the authenticity of the work.

See if there is a history of AI-generated text in the student's work. We recommend looking for a long-term pattern of AI use, as opposed to a single instance, in order to determine whether the student is using AI.