SCA Strikes Back: Reverse Engineering Neural Network Architectures using Side Channels

Lejla Batina[1], Shivam Bhasin[2], Dirmanto Jap[2], Stjepan Picek[3]
[1]Faculty of Science, Radboud University,
Postbus 9010, 6500 GL Nijmegen, Netherlands.
[2]Temasek Laboratories, Nanyang Technological University,
50 Nanyang Drive, Research Techno Plaza, BorderX Block, 9th Storey, Singapore 637553
[3]Delft University of Technology,
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands.
Email: lejla@cs.ru.nl, {djap,sbhasin}@ntu.edu.sg, picek.stjepan@gmail.com

Machine learning, and more recently deep learning, have become mainstream research and development directions due to their unquestionable practicality and effectiveness. The ever-increasing computational capabilities of modern computers and huge amounts of available data result in ever more complex and effective machine learning architectures. Deep learning algorithms gain popularity in edge devices such as sensors or actuators, as they are indispensable for real time processing. Consequently, there is an increasing interest in deploying neural networks on low-power processors found in always-on systems like ARM Cortex-M microcontrollers. It is expected that by 2024, the number of edge-based artificial intelligence chips to be doubled [1].

With the increasing number of design strategies and machine learning algorithms to use, fine-tuning algorithm's hyperparameters is emerging as one of the main challenges. The design and training of a machine learning model is a challenging procedure and an expensive one, so a well-trained model has a monetary value. For instance, the cost of training a machine learning model can be more than one million USD [2].

We are also witnessing an increase in intellectual property (IP) model strategies. In cases when optimized neural networks are of commercial interest, their details must be kept undisclosed. IP thefts of trained machine learning models through side-channel attacks are becoming a major threat. Setting aside privacy issues, obtaining valuable information from neural network architectures can help acquire trade secrets from the competition, leading to losses in competitive advantage. Despite the advantages of using machine learning-enabled edge devices, it becomes harder to ensure the confidentiality of the developed model as the devices operate in an environment where physical side-channels analysis becomes a real threat.

This work was originally published in USENIX Security 2019 [3]. Since then, several new results have been published, inspiring this line of research on side-channel and fault attacks on neural networks. Prior to this work, reverse engineering attacks on neural networks mostly relied on observing the outputs of the neural network and training a substitute model or exploiting specific design choices. This work shows that it is possible to recover the layout of unknown neural networks by exploiting the available physical (side-channel) information. Our approach does not need access to training data and allows for neural network recovery by feeding known random inputs.

I. Background

A. Machine Learning

Machine learning, in general, is based on the idea that a system can "learn" from examples by extracting patterns or discovering information without human intervention [4]. There are many different machine learning algorithms. Today, the most popular algorithms come from the neural network family and are based on the deep learning paradigm.

B. Side-channel Analysis

Side-channel analysis (SCA) exploits the vulnerabilities of implementations. It was first demonstrated on cryptographic implementations [5]. SCA shows that even for theoretically secure algorithms, observing the unintentional physical or side-channel leakages (such as timing, power, electromagnetic emanation) from their implementations, could lead to the potential recovery of secret information. Next, we describe some of the most common methods used in SCA, which we will also use in our attacks. We discuss timing analysis, Simple Power Analysis (SPA), and Differential Power Analysis (DPA) [1].

Timing Analysis. When the algorithm is implemented, different operations lead to different timing execution. If the execution time depends on sensitive parameters, it leaks sensitive information to the adversary. In our attack, we exploit the unique timing behavior of various activation functions.

Simple Power Analysis (SPA). In SPA, one learns sensitive information from one or a few traces, with basic techniques like a visual inspection supported by signal processing. In the context of neural network reverse engineering, SPA can determine the number of neurons and even the number of layers in some cases.

Differential Power Analysis (DPA). The attack applies statistical techniques for the secret recovery. The general idea is

[1]Note that despite the attack name (power analysis), it could also be used on another side-channel leakage, such as the electromagnetic (EM) emanation.

to test or identify statistical dependencies between the physical leakage and the hypothetical intermediate value (secret dependent). For example, the adversary could compute a (nonlinear) function between the known value and hypothetical secret. The adversary then applies a leakage model on the output, which is generally device-dependent (e.g., the Hamming weight and Hamming distance model). Statistical tests are then used to compare different hypothetical values (influenced by the different hypotheses of the secret) with the physical leakage. The most commonly used statistical method is correlation, as used e.g. in Correlation Power Analysis (CPA). It computes the Pearson correlation between each hypothetical output and the physical leakages. The hypothetical secret that leads to the highest absolute correlation value is then deemed the right guess. We use CPA to recover the secret weights as well as to determine the layer boundaries.

II. Model Recovery Techniques Overview

This section provides a brief introduction to the machine learning model recovery attack in embedded devices using electromagnetic side-channel. Interested readers are referred to [3] for extensive technical details of the attack process.

A. Threat Model

The threat model for the attack assumes an adversary interested in recovering the architecture (hyperparameters) and parameters of the target model. The target is a pre-trained neural network model executed on an embedded device while running inference. The adversary can query the model with known/chosen inputs and passively observe side-channel information corresponding to the executed inference. For the following experiment, we observe electromagnetic side-channel signatures, thus requiring physical access to the device. While most model extraction attacks need access to the original training dataset (or similar dataset), the attack proposed in the following does not need access to training data. As shown later, an adversary can feed random known inputs to extract the model. To be as generic as possible, we work with randomly chosen real numbers as inputs. Finally, the target model is assumed to have no side-channel countermeasures implemented, which is (unfortunately) true almost everywhere in practice today.

B. Experimental Setup

The experimental setup comprises of the target embedded device (e.g., 8-bit Atmel ATmega328P and 32-bit ARM Cortex-M3), executing the model, an electromagnetic (EM) probe to monitor side-channel activity, a digital oscilloscope to capture measured side-channel activity, and an optional pre-amplifier to boost the measured signal. The side-channel activity is captured using the Lecroy WaveRunner 610zi oscilloscope using an RF-U 5-2 near-field electromagnetic (EM) probe from Langer and a 30dB pre-amplification. We use available handshaking signals like the start/stop of computation to synchronize the measurements. Each measurement (or trace) corresponds to one randomly chosen input. Every trace

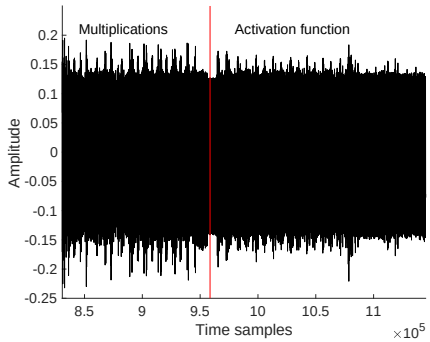


Fig. 1: Observing pattern and timing of multiplication and activation function.

is composed of several samples (or points), where the number of samples can go in the range of millions when measuring a complete inference. Since we use a microcontroller, the neurons are executed sequentially. The attack targets leakage corresponding to the loading of sensitive parameters in the data bus, which is known to leak with the Hamming Weight (HW) model, i.e., proportional to the number of bits equal to '1' in the sensitive variable. [5]. The target models are implemented in C language and pre-trained offline.

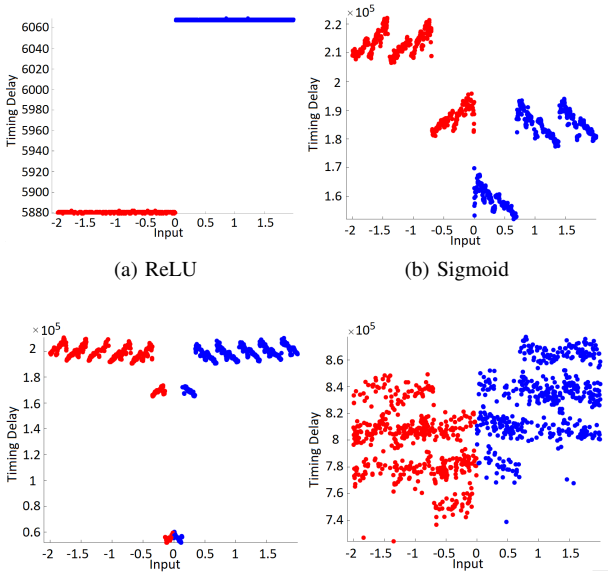
C. Recovering Neural Network Parameters

First, we implement a simple Multilayer Perceptron as a toy example. Multilayer Perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. It consists of multiple layers of nodes in a directed graph. Each node in a layer is connected to every node in the subsequent layer, and each connection is associated with a certain weight parameter.

The implemented architecture consists of one hidden layer with six neurons. Each neuron implements the input multiplication followed by the Sigmoid activation function. The execution sequence as captured on the side-channel trace is shown in Figure 1. Notice that the multiplication and activation are clearly distinguishable (separated by the red line for readability).

1) Recovering Activation Function: Activation functions are the main nonlinear component of a neural network [4].

As the activation function is clearly distinguishable on the captured EM trace, one can easily measure the timing execution from the EM trace and be precise to a nanosecond scale. We observed that all activation functions have a unique timing behavior, which leaks information about the function used. We analyze the timing behavior of four commonly used activation functions: ReLU, Sigmoid, Tanh, and Softmax. The timing behavior for 2 000 random inputs is shown in Figure 2 and allows distinguishing each activation function. To recover activation functions for the whole network, an adversary feeds random inputs and records the execution timing for each activation function in each neuron. For a modern oscilloscope, side-channel activity for all the neurons can be captured at



(a) ReLU (b) Sigmoid

(c) Tanh (d) Softmax

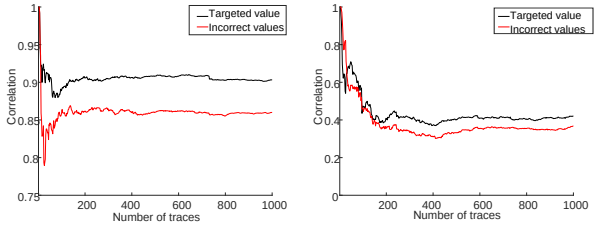
Fig. 2: Timing behavior for different activation functions.

once for one input, and the same traces can be used to recover activation functions for the whole network.

2) Recovering Neural Network Weights: The weights of a pre-trained model make the core of the intellectual property. In many cases, the architecture might be known publicly, but it is the weights resulting from detailed training that distinguishes a good model from a bad one. We target weight recovery with CPA. It is assumed that the adversary can synchronize the weight multiplication from one input to another, using widely available techniques in the side-channel literature [5].

The attack targets multiplication of secret weight w with i^{th} known input x_i, resulting in product p_i. The leakage occurs when p is computed and stored back in the memory. While the implementation of multiplication can vary (schoolbook, software optimized, hardware-accelerated), the storage of p will leak, and thus it is easier to target it. On the microcontroller, writing p to memory follows the HW leakage. Thus, the CPA computes Pearson correlation $\rho(t, HW(p))$ for all hypotheses of w, corresponding to a set of inputs x. Here, t represents the set of side-channel traces captured corresponding to inputs x. Given a sufficient number of traces, the correlation for correct w will stand out from other wrong hypotheses. This is analogous to secret key recovery in cryptography, where the HW leakage of a key-dependent intermediate value is targeted for known plaintext to find the secret key with the highest correlation. Still, there is an important difference. In cryptography, we require exact key recovery, but here, some precision errors can be tolerated.

The underlying implementation treats weights in IEEE 754 representation, where each weight is represented in 32 bits. The most significant bit represents the sign, the next eight bits contain the exponent, and the remaining 23 are reserved for the mantissa. We recover them as four independent bytes in four independent attacks. The traces remain the same as they all correspond to the same multiplication, only our hypothesis changes when moving from one byte to other. Of course, the first two bytes are more important, comprising of sign, exponent, and most significant mantissa bits. The attack on the first two bytes is shown in Figure 3. The black line represents the correlation with the correct weight and the red lines for incorrect weight. The y-axis represents absolute correlation and the number of traces (or corresponding inputs queried) on the x-axis. The attack is considered successful when the black line depicts a higher correlation over the red line in a conclusive manner. With around 200 traces, the correct weight can be identified. The same attack must be repeated on each multiplication to recover other weights.



(a) First byte recovery (sign and 7-bit exponent) (b) Second byte recovery (LSB exponent and mantissa)

Fig. 3: Recovery of weights in a neural network.

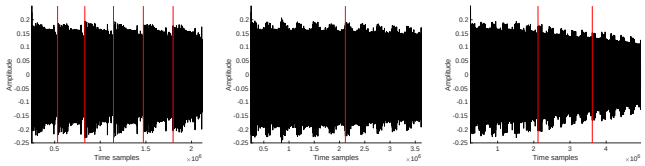
D. Recovering Neural Network Architecture

Once the weights and activation functions are recovered, only the architecture remains to be recovered. This is performed using Simple Power Analysis (SPA), which relies on visual inspection of side-channel measurements to learn sensitive information.

We noticed that the neurons have a very distinct side-channel signature in a sequential execution setup like ours. Consider Figure 4, which shows execution signature of three neural networks with (6), $(6, 5)$, and $(6, 5, 5)$ architectures. Here (a, b, c) represents a feed-forward neural network with three hidden layers and a, b, c neurons in each layer, respectively, starting from the input layer. As shown in Figure 4, the number of neurons can be easily recovered with SPA. Layer boundaries are not clear by SPA, and CPA is used for that purpose. Here, CPA exploits the fact that neurons in the first layer will show a higher correlation with the inputs than the second and later layers, allowing the identification of neurons in the first layer. The boundaries of different layers can be determined similarly.

III. Evaluation

A combination of previously discussed techniques recovers the full neural network. The recovery is performed layer by layer, neuron by neuron. The recovery of the previous layer allows the adversary to compute inputs to the next layer and continue the attack to recover the weights and structure. The



(a) (6) (b) (6,5) (c) (6,5,5)

Fig. 4: SPA on hidden layers.

methodology to reverse engineer a neural network is displayed in Figure 5. This methodology scales linearly with the size of the neural network.

A. Reverse Engineering MLP

We consider an MLP with $(50, 30, 20, 50)$ architecture that was previously used for side-channel applications in [6]. This neural network is implemented in ARM Cortex-M3 as it allows testing our approach with considerably larger neural network models than discussed up to now. All the activation functions are ReLU except the output layer, which uses Softmax. The measurement trace is shown in Figure 6(a). The dataset is DPAcontest v4 with 50 samples and 75 000 measurements where the first 50 000 measurements are used for training and the rest for testing. The dataset has nine classes.

The four layers and their boundaries are clearly distinguishable. Moving forward, we show the measurement for one neuron in the third layer in Figure 6(b), where 20 multiplication peaks and ReLU peaks are visible. We performed the neural network model extraction with the previously described approach. The recovered model has an accuracy of 0.6087, compared to 0.6090 for the original model.

B. Reverse Engineering CNN

We finally extend the proposed attack methodology to Convolutional Neural Networks (CNNs). CNNs are inspired by the biological processes of animals' visual cortex to process data with 2-dimensional convolutions. CNNs are mainly composed of convolutional layers, pooling layers, and fully connected layers. Convolutional layers are linear layers that share weights across space. Pooling layers are nonlinear layers that reduce the spatial size to limit the number of neurons. Fully connected layers are layers where every neuron is connected with all the neurons in the neighborhood layer.

The target is the CMSIS-NN implementation on ARM Cortex-M3 with the same measurement setup as in previous experiments. As input, we target the CIFAR-10 dataset that consists of 60 000 32×32 color images in 10 classes. The CNN consists of three convolutional layers, three max-pooling layers, and one fully connected layer. We choose as target the multiplication operation from the input with the weight, similar as before. For this experiment, the operations on real values are performed using fixed-point arithmetic.

For the pooling layer, once the weights in the convolution part are recovered, the output can be calculated. Since the max-pooling layer is based on the following conditional instruction, $conditional(if(a > max)\ max = a)$, it is straightforward to differentiate it from the average pooling that has summation and division operations. This technique is then repeated to reverse engineer any number of convolutional and pooling layers. Finally, the fully connected layer is recovered in the same way as done for MLP. In our experiment, the original accuracy of the CNN equals 0.7847, and the accuracy of the recovered CNN is 0.7811.

IV. Perspectives and Long-term Impact

Physical attacks on machine learning and deep learning implementation have received growing interest from the research community. While this work [3] is one of the first works highlighting physical channel vulnerabilities on deep learning, it was validated on microcontrollers only. Nevertheless, it has motivated several directions for further research.

A natural question arises regarding the feasibility of such attacks on other hardware platforms. Dubey et al. [7] presented the first practical model recovery attack on FPGA platforms, followed by a proposal to integrate masking as a countermeasure. Recently, it was also shown that model recovery attacks could also be performed remotely on multi-tenant FPGA [8], thus relaxing the requirement for physical access. Attacks on neural networks not only threaten the recovery of confidential models but also the sensitive input can be recovered with a similar approach [9]. Chmielewski and Weissbart managed to reverse engineer implemented neural network on Nvidia Jetson Nano, a module computer embedding a Tegra X1 SoC combining an ARM Cortex-A57 CPU and a 128-core GPU within a Maxwell architecture by using simple EM analysis [10]. Further, side-channel in a server setting has threatened cloud-based model execution, as demonstrated by Wei et al. [11]. A comprehensive survey of SCA-based model recovery attacks is presented in [12].

The threat of model extraction attacks on neural networks has also driven prompt action from the industry. Vendors of neural network accelerators like Intel and Nvidia also now include features for model protection. Intel, under its *OpenVINO* framework, recommends the use of secure enclaves for sensitive model execution and provides features like model encryption. Several security add-ons features are available for vendors to enable the creation, distribution, and application of models in a secure setting. Nvidia, with their latest *EGX100* platforms, have introduced the concept of *Confidential AI enclaves* to prevent IP theft. With the highlighted vulnerability from [3] and follow-up action from both academia as well as industry, the effort to protect sensitive machine learning models has gain momentum. Alongside, we also motivate research in solving these vulnerabilities with a holistic approach under the security by design paradigm.

V. Discussion

Our previous work [3] selected for *Top Picks in Hardware and Embedded Security 2020* demonstrates that it is possible to reverse engineer neural networks by using side-channel attacks. We developed a framework that considers each part

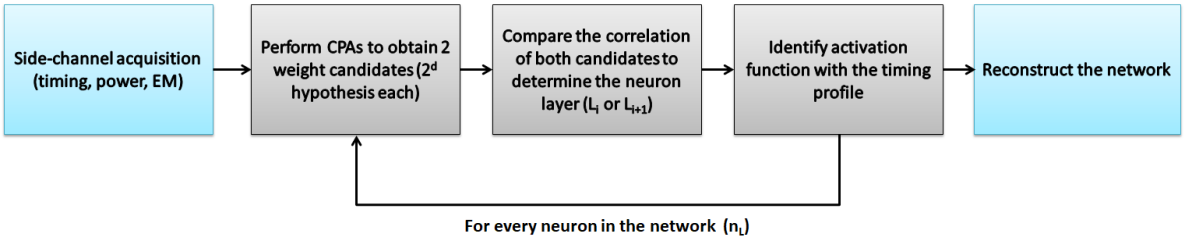
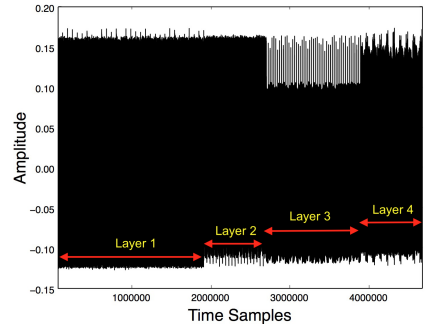
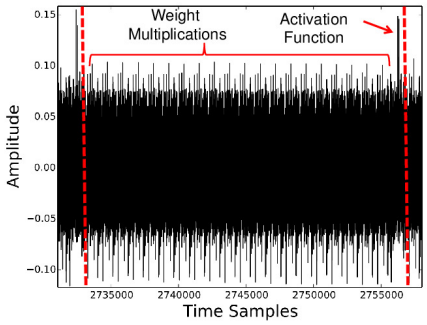


Fig. 5: Methodology to reverse engineer a neural network.



(a) Full EM trace for MLP $(50, 30, 20, 50)$



(b) EM trace for one neurons in 3^{rd} layer of MLP $(50, 30, 20, 50)$

Fig. 6: Reverse engineering neural network on ARM Cortex-M3 with (a) showing full execution trace allowing identification of each layer and (b) showing a zoom-in at one neuron execution in third layer with expected 20 multiplication peaks followed by a ReLU execution peak.

of the neural network separately and then, by combining the information, manages to reverse engineer all relevant hyperparameters and parameters. Our work is a proof of concept (but also a realistic demonstration) that such attacks are possible and warns that more effort should be given to developing countermeasures. While we have used microcontrollers for our experiments, the attack applies to other targets like FPGAs and GPUs.

Acknowledgment

This research is partly supported by the National Research Foundation, Singapore, under its National Cybersecurity Research & Development Programme / Cyber-Hardware Forensic & Assurance Evaluation R&D Programme (Award: NRF2018NCR-NCR009-0001).

References

[1] D. Insights, "Bringing AI to the device: Edge AI chips come into their own," 2020, https://www2.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/2020/ai-chips.html.

[2] E. Strubell, A. Ganesh, and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP," *CoRR*, vol. abs/1906.02243, pp. 1–6, 2019. [Online]. Available: http://arxiv.org/abs/1906.02243

[3] L. Batina, S. Bhasin, D. Jap, and S. Picek, "{CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 515–532.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[5] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[6] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni, "The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 1–29, 2019.

[7] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 197–208.

[8] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Power Side-Channel Attacks on BNN Accelerators in Remote FPGAs," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2021.

[9] L. Batina, S. Bhasin, D. Jap, and S. Picek, "Recovering the Input of Neural Networks via Single Shot Side-channel Attacks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2657–2659.

[10] L. Chmielewski and L. Weissbart, "On reverse engineering neural network implementation on GPU," in *Applied Cryptography and Network Security Workshops - ACNS 2021 Satellite Workshops*, ser. Lecture Notes in Computer Science, J. Z. et al, Ed., vol. 12809. Springer, 2021, pp. 96–113. [Online]. Available: https://doi.org/10.1007/978-3-030-81645-2_7

[11] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque, "Leaky DNN: Stealing Deep-Learning Model Secret with GPU Context-Switching Side-Channel," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 125–137.

[12] H. Chabanne, J.-L. Danger, L. Guiga, and U. Kühne, "Side channel attacks for architecture extraction of neural networks," *CAAI Transactions on Intelligence Technology*, vol. 6, no. 1, pp. 3–16, 2021. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12026

Lejla Batina is a professor in embedded systems security at the Radboud University, Netherlands. She received her PhD. from KU Leuven (2005) and she has also worked as a cryptographer in industry. She is a senior member of IEEE and an Editorial board member of some top journals in security.

Shivam Bhasin is a Senior Research Scientist and Programme Manager (Cryptographic Engineering) at NTU Singapore. He received his PhD from Telecom Paristech (2011) and Master's from Mines Saint-Etienne (2008). Before NTU, Shivam held position of Research Engineer in Institut Mines-Telecom, France. His research interests include embedded security and trusted computing.

Dirmanto Jap is a Research Scientist at PACE Lab, Temasek Laboratories, Nanyang Technological University (NTU), Singapore. He previously received his PhD in Mathematics from NTU in 2016. His main research topics include physical attacks and countermeasures, practical fault injection, application of machine learning for security.

Stjepan Picek is an assistant professor at the Delft University of Technology, The Netherlands. He received his PhD in 2015, and from 2015 to 2017, he was postdoctoral research at KU Leuven, Belgium and MIT, USA. His research interests include security, machine learning, and evolutionary algorithms.