On Comparing Side-channel Properties of AES and ChaCha20 on Microcontrollers

Zakaria Najm[1,2], Dirmanto Jap[1], Bernhard Jungk[3], Stjepan Picek[2], and Shivam Bhasin[1]
[1]*Temasek Laboratories, Nanyang Technological University, Singapore*
[2]*Delft University of Technology, Delft, The Netherlands*
[3]*Independent Researcher*
{zakaria.najm,djap,sbhasin}@ntu.edu.sg, bernhard@projectstarfire.de, s.picek@tudelft.nl

***Abstract*—Side-channel attacks are a real threat to many secure systems. In this paper, we consider two ciphers used in the automotive industry – AES and ChaCha20 and we evaluate their resistance against side-channel attacks. In particular, the focus is laid upon the main non-linear component in these ciphers. Owing to the design of ChaCha20, it offers natural timing side-channel resistance and thus is suitable for affected applications. However, attacks exploiting the power side-channel are somewhat more difficult on ChaCha20 as compared to AES, but the overhead to protect ChaCha20 against such attack is considerably higher.**

***Index Terms*—Side-Channel Attacks, Machine learning, Cryptography, Automotive**

I. Introduction

The automotive industry has seen a major digital transformation in recent years. Nowadays, electronics can make up to 60-70% of a car's development cost and consequently, many electronic control units (ECUs) communicate with each other or to external units, to monitor and manage tasks of various criticality. Any manipulation of the communicated data can have a direct effect on the functional safety of the vehicle and thus, can lead to serious consequences. To protect against such threats, suites of standard cryptographic algorithms are used to ensure the integrity of the data.

For securing the data, standards for symmetric encryption like AES [1] and ChaCha20 [2] are used in the automotive industry. While many lightweight ciphers have been developed in recent years, only AES and ChaCha20 are usually used in automotive application. One of many threats for automotive applications are side-channel attacks (SCA [3]), which target implementations of cryptographic algorithms and lead to key recovery in negligible computation time. While AES is a standard for the last two decades, ChaCha20 is seeing a rapid adoption due to its software-friendly design and natural resistance to timing side-channel attacks.

AES uses a substitution box (S-box) as the only nonlinear component, which can be either pre-computed or calculated on the fly which involves the inversion in the Galois field $\mathrm{GF}(2^8)$. ChaCha20 uses a modular addition instead, which is natively supported on a wide range of general purpose microcontrollers. Since SCA primarily targets the nonlinear component, these two operations become the focus of our study. To perform a detailed analysis, we use profiled machine learning attacks and non-profiled correlation power analysis, some of the most commonly used SCA techniques.

When considering countermeasures, it is also the nonlinear operation which is expensive to implement protected against side-channel attacks, unlike other linear components of the cipher. Thus, we also study and compare the cost of protecting these two basic functions (S-box and modular addition) against SCA. Other physical attacks like fault attacks and hardware Trojan are considered out of scope due to different attack model and modus operandi.

The rest of the paper is organized as follow. Section II recalls basics of target algorithms and side-channel attacks. Section III compares the susceptibility of the AES S-box and the modular addition to profiled machine learning-based attacks and non-profiled correlation power analysis. In this paper, we use simulated measurements for the analysis. Section IV analyzes the two nonlinear operation in terms of the side-channel protected implementation overhead. Finally, conclusions are drawn in Section V.

II. Background

A. Target Algorithms

1) AES [1]: The Advanced Encryption Standard (AES) is the NIST standard for block ciphers [4]. AES follows the Substitution Permutation Network (SPN) construction and operates on 128 bit data blocks with a 128/192/256 bit secret key in 10/12/14 rounds [1]. Four distinct operations comprise a round, i.e., SubBytes (SB), ShiftRows (SR), MixColumns (MC), and AddRoundKeys. The operations are applied on the 128-bit data state that is organized as a 4×4 matrix of bytes. SubBytes is a non-linear permutation on individual bytes. ShiftRows and MixColumns are linear permutations of the state and AddRoundKeys performs exclusive-or (XOR) between round keys and the data state. A key expansion algorithm is used to derive round keys from the master key. All rounds are identical except the last one, which skips MixColumns. The SubBytes operation is based on the computation of the inverse of an element of $\mathrm{GF}(2^8)$, followed by an affine transformation. Often, the computation is performed using a precomputed table. The implementation of an S-box is not easy and often represents the main source of a large time or memory footprint. Hardware support for AES is currently available in a large range of processors from vendors like Intel, AMD, or ARM. With the built-in support, recent applications have seen significant performance improvements.

2) ChaCha20 [2]: The ChaCha20 stream cipher is a part of ChaCha20-Poly1305 Authenticated Encryption with Associated Data (AEAD) suite. It follows the ARX construction, i.e., it is based only on addition, rotation, and XOR. Its design is optimized for efficiency and is easily implemented as a time-constant software. It produces a key stream with the internal state initialized using 4×32 bit constant values $c_0, \ldots, c_3$, an 8×32 bit width secret key $k_0, \ldots, k_7$, 3×32 bit nonce values n_0, n_1, n_2, and a 32 bit message block counter value count. Then, the complete state is copied, before the double-round function is called 10 times. The plaintext is XORed with the key stream to get the ciphertext. The only nonlinear function in ChaCha20 is the modular addition, which is very efficient in both hardware and software. Being fairly new, currently no hardware support is available, but ChaCha20 is anyhow adopted in many commonly used cryptographic libraries.

B. Side-Channel Attacks and Metrics

1) Profiled Side Channel Attacks: Profiled SCA assumes a strong attacker who has access to a clone device allowing detailed characterization of the leakage. While earlier proposals used template attacks [5], machine learning classifiers were shown to outperform template attacks in certain scenarios [6].

Random Forest (RF) is a well-known ensemble decision tree learner [7]. Decision trees choose their splitting attributes from a random subset of k attributes at each internal node. The best split is taken among these randomly chosen attributes and the trees are built without pruning, RF is a parametric algorithm with respect to the number of trees in the forest. Random Forest is a stochastic algorithm because of its two sources of randomness: bootstrap sampling and attribute selection at node splitting. Learning time complexity for RF is approximately $\mathrm{O}\big(I \cdot k \cdot NlogN\big)$. We use $I = [10, 50, 100, 200, 500, 1\,000]$ trees in the tuning phase, with no limit to the tree size. To evaluate the performance of RF, we use accuracy as the metric. Accuracy is defined as:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}. \tag{1}$$

Here, TP refers to true positive (correctly classified positive), TN to true negative (correctly classified negative), FP to false positive (falsely classified positive), and FN to false negative (falsely classified negative) instances.

We divide the traces into training and testing sets in a 2:1 ratio. On the training set, we conduct a 5-fold cross-validation where we use the averaged results of individual folds to select the best classifier parameters. We report results from the testing phase only, as these results are more relevant than the training set results in assessing the actual classification performance of the constructed models. All the machine learning experiments are done with the scikit-learn library [8] for Python.

2) Non-Profiled Side Channel Attacks: Non-profiled SCA assumes a weaker attacker who has no access to a clone device, but can observe side-channel activity for known inputs or outputs. Based on a leakage model (like the Hamming Weight or the Hamming Distance), the attacker can estimate the leakage based on key hypotheses. Next, statistical means like correlation are used to find a dependency between the observed leakage and the estimated leakage. The correlation maximizes for the estimated leakage with correct key, thus revealing its value to the attacker. We use the Pearson correlation coefficient as the statistical distinguisher. The divide and conquer approach allows to treat small parts of the key (for example, one byte) at a time, making the complexity practical for both profiled and non-profiled SCA.

3) Attack Metric: We use guessing entropy as the attack metric. The Guessing entropy measures the average number of key candidates to test after the attack. A guessing entropy of 1 signifies a successful attack. The Guessing entropy of the adversary $A_{E_k,L}$ against a key class variable S is defined as:

$$GE_{A_{E_K,L}}(\tau, m, k^*) \quad = \quad \mathsf{E}[\mathrm{Exp}_{A_{E_K,L}}].$$

III. Side-channel Analysis of Target Algorithms

In this section, we analyze and compare the side-channel susceptibility of the basic nonlinear functions of AES and ChaCha20. To perform the worst case analysis, we consider the strongest attacker with the profiling capability. Further, we use a machine learning classifier for the attack as such techniques have shown to perform better than template attacks in certain cases [6].

We consider a 32 bit microcontroller as our target as it is seeing rapid adoption in automotive applications. The simulated device is considered to be leaking in the perfect Hamming weight model with some added environmental (Gaussian) noise of standard deviation σ. Two levels of noise are tested. In one experiment, the AES S-box is implemented as a lookup table in memory, operating over 8-bits. Although the underlying architecture is 32 bit, due to memory restrictions, an 8 bit lookup is normally used and thus, we consider only S-box look-ups over 8 bit in our analysis.

For the case of the modular addition, our simulation is computed over 32 bits owing to the native support in the instruction set. An attack over 32 bits might be sometimes slow on normal system. However, the attacker has the freedom to observe it over any granularity from 1 bit to 32 bit. This is represented as (x/y/z) in Table I. Here, x signifies the bit width of observation or the side-channel signal, while the remaining of 32 bit ($y = 32 - x$) contribute to algorithmic noise. $z = x + 1$ is the total number of classes in the attack.

Our results are shown in Table I. With the low environmental noise and no algorithmic noise, the attack accuracy is 100%. For other cases, the accuracy reduces with increased noise and increased number of classes (z). In general, modular addition is marginally harder to attack as compared to the AES S-box, because of the higher nonlinearity of the latter. Note that the accuracy for scenario $(32/0/33)$ has 100% accuracy when $\sigma = 0.1$, which is much higher than for $(16/16/17)$ case. Although this may sound counterintuitive, since the number of classes is much higher in the first case, there is a simple explanation of such a behavior. When the level of

TABLE I
CLASSIFICATION ACCURACY (%)

Noise	AES	ChaCha20			
σ	(8/0/9)	(4/28/5)	(8/24/9)	(16/16/17)	(32/0/33)
0.1	100	30.19	23.02	20.04	100
1	34.01	29.50	22.43	18.78	29.58

noise is small, there are clear boundaries between classes and RF does not have any problems when classifying. However, when we reduce the number of classes (granularity), then we actually see an overlap among certain measurements since now they can belong to several classes, which results in a wrong classification.

In addition to the accuracy metric, we use the guessing entropy as a metric for the SCA evaluation. In the attack scenario, the attacker is more interested in obtaining the correct secret key, rather than exact classification. The attacker can easily calculate the maximum likelihood for each key candidate, and derive the guessing entropy, i.e., the average rank of the correct key.

In Table II, the guessing entropies for RF are presented. It shows that for $\sigma = 0.1$, with 2^{10} traces, it is enough to recover the correct key, whereas for $\sigma = 1$, it will require 2^{11} traces. In general, it can be observed that even though some of the labels are not predicted correctly, by taking the maximum likelihood, it is still possible to recover the secret key.

When considering non-profiled SCA, the results are presented in Table III. The attack model here is quite different from the profiled attacks, thus a direct comparison is not possible. However, it can be clearly seen that in both cases, addition offers slightly higher resistance to SCA as compared to the S-box.

IV. TOWARDS SIDE-CHANNEL PROTECTION

A. Preventing Timing Side-channels

In most cases, timing side-channels can be prevented by removing data-dependent branches and memory accesses. For ChaCha20, this is trivial, because no operation in the algorithm uses operations that are susceptible to such behavior [2].

However, a fast AES implementation uses lookup tables to implement either the S-boxes or so-called T-tables. Such table lookups usually do not have a constant time behavior, and thus, a naive AES implementation is vulnerable to cache timing attacks [9]. Using bit-slicing, the data-dependent behavior can be removed, but at the cost of a reduced performance. An alternative countermeasure is to disable caches during the execution of AES, if possible.

Another possibility is to use hardware accelerators that many modern microcontrollers implement. For AES, this results in a much faster, constant-time computation. The downside is, that many AES accelerators are not resistant to power side-channel attacks and therefore, a software implementation is needed, if such a protection is required.

No modern microcontroller has a ChaCha20 accelerator so far and therefore, no fair comparison is possible. However, due to the construction of ChaCha20 as an ARX cipher, we believe that at least for low-end and mid-end processors, it is unlikely that accelerators will be developed. For high-end processors the usage of already existing SIMD instructions can lead to high speed ups.

When we look at our main target platforms, i.e., ARM, most of the time AES and ChaCha20 perform with similar speed. However, if hardware acceleration for AES is available, it becomes a significant advantage for AES.

B. Preventing Power Side-channels

Power side-channels are much harder to mitigate, because the overhead of protected implementations is much higher than for the protection against timing side-channels.

One of the most effective countermeasures against power analysis attacks is masking. In software, many masking schemes of nonlinear gates are often based on the Trichina gate [10] or optimized variants [11].

The common factor among all masking countermeasures is their high overhead compared to implementations hardened against timing side-channels. For our comparison, we outline the costs for masking ChaCha20 and AES.

For ARX ciphers such as ChaCha20, there are two possible approaches. First, it is possible to mask the addition operation with low overhead with arithmetic masking and apply Boolean masking to linear operation. The drawback of this approach is a costly conversion between the different types of masks [12]. Second, a Boolean masking can be applied to all operations. Then, instead of having an expensive mask conversion step, masking of the addition itself becomes expensive [13]. Currently, the most efficient approach for Cortex-M3/M4 ARM microcontrollers needs 78 instructions for one addition [14]. Therefore, it is necessary to spend at least $16 \times 21 \times 78 = 26\,208$ instructions to mask all addition operations of ChaCha20.

For AES, Boolean masking is usually implemented in a very different way. Schwabe et al. [15] proposed a bit-sliced approach to implement the SubBytes layer, which needs 688 instructions for each SubBytes layer. Since AES has 10 rounds, we only need to spend $6\,880$ instructions to implement the SubBytes layer of AES. A similar bit-slicing optimization is less beneficial for ChaCha20, because the operations used in the algorithm presented in [14] are already working with 32 bit in parallel. Therefore, the hardware utilization cannot be improved significantly by reorganizing the internal state.

Of course, we also need additional instructions for the linear operations of both ciphers. However, With the exception of added loads and stores, the amount of instructions only grows linearly with the protection order. Therefore, AES seems to be much faster, when protections against power side-channels are needed. Currently, the most efficient first-order protected implementation of AES-128 and ChaCha20 known in the literature take 463.9 [15] and 947.2 [14] clock cycles per byte, respectively, if long messages are processed. However, for short messages, such as those typically transmitted over the CAN bus, ChaCha20 is much worse than AES, since the

TABLE II
GUESSING ENTROPY FOR RF

σ: 0.1												
	No of traces (2^x)											
	1	2	3	4	5	6	7	8	9	10	11	12
S-box	5.3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 4	10.0	6.7	6.3	3.0	2.0	2.0	1.7	2.0	1.0	1.0	1.0	1.0
Add 8	181.0	126.5	66.3	76.1	48.6	42.0	5.0	3.0	1.3	1.0	1.0	1.0
Add 16	21 075.8	12 505.3	23 298.3	11 935.6	809.7	37.3	103.0	4.3	3.0	1.0	1.0	1.0
Add 32	1 846.0	35.7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
σ: 1												
	No of traces (2^x)											
	1	2	3	4	5	6	7	8	9	10	11	12
S-box	84.0	20.1	10.3	8.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 4	11.0	6.7	6.7	12.0	5.6	7.0	8.0	2.0	1.3	1.0	1.0	1.0
Add 8	134.0	69.0	36.2	49.7	38.0	57.0	86.0	7.6	16.0	3.0	1.0	1.0
Add 16	33 926.0	23 171.3	32 294.7	14 162.7	4 512.0	11 324.3	688.3	11.0	1.0	1.0	1.0	1.0
Add 32	14 305.3	3 134.3	5 736.3	22.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

TABLE III
GUESSING ENTROPY FOR CPA

σ: 0.1												
	No of traces (2^x)											
	1	2	3	4	5	6	7	8	9	10	11	12
S-box	155.0	3.3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 4	5.7	10.0	15.0	10.3	5.6	2.3	1.0	1.0	1.0	1.0	1.0	1.0
Add 8	180.7	67.0	18.6	85.0	8.7	2.0	1.3	1.0	1.0	1.0	1.0	1.0
Add 16	40753.0	10 288.0	2 501.0	34.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 32	42 121.0	9.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
σ: 1												
	No of traces (2^x)											
	1	2	3	4	5	6	7	8	9	10	11	12
S-box	184.4	56.7	35.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 4	13.0	12.0	7.3	3.6	6.4	2.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 8	226.0	104.0	71.1	9.7	9.0	1.3	1.0	1.0	1.0	1.0	1.0	1.0
Add 16	40 644.0	4 819.0	15 098.0	1 349.0	112.0	3.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 32	43 313.0	1 840.0	114.0	6.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

minimum block size is 512 bit, while AES processes only 128 bit blocks, hence a significant slowdown is expected.

V. CONCLUSIONS

We compare the AES S-box and the modular addition used by ChaCha20, the main nonlinear components of two cryptographic algorithms regarding their side-channel vulnerability. While the addition operation is slightly harder to attack as compared to the S-box, the overhead for protecting an addition is much higher. Thus, if power side-channel attacks are not a concern, like in remote applications, ChaCha20 may have some advantages over AES owing to its software performance and natural resistance to timing attacks. However, if resistance against power side-channel attacks is required, the currently best known implementations favor AES.

REFERENCES

[1] V. Rijmen and J. Daemen, "Advanced Encryption Standard," *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology*, pp. 19–22, 2001.

[2] Y. Nir and A. Langley, "Chacha20 and poly1305 for ietf protocols," Tech. Rep., 2018.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[4] N.-F. Standard, "Announcing the Advanced Encryption Standard (AES)," *Federal Information Processing Standards Publication*, vol. 197, pp. 1–51, 2001.

[5] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 13–28.

[6] S. Picek, A. Heuser, A. Jovic, S. A. Ludwig, S. Guilley, D. Jakobovic, and N. Mentens, "Side-channel analysis and machine learning: A practical perspective," in *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, 2017, pp. 4095–4102.

[7] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

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

[9] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *Cryptographers Track at the RSA Conference*. Springer, 2006, pp. 1–20.

[10] E. Trichina, "Combinational Logic Design for AES SubByte Transformation on Masked Data," *IACR Cryptology ePrint Archive*, vol. 2003, p. 236, 2003.

[11] A. Biryukov, D. Dinu, Y. Le Corre, and A. Udovenko, "Optimal First-Order Boolean Masking for Embedded IoT Devices," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2017, pp. 22–41.

[12] J.-S. Coron and A. Tchulkine, "A new algorithm for switching from arithmetic to boolean masking," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2003, pp. 89–97.

[13] J.-S. Coron, J. Großschädl, M. Tibouchi, and P. K. Vadnala, "Conversion from arithmetic to boolean masking with logarithmic complexity," in *International Workshop on Fast Software Encryption*. Springer, 2015, pp. 130–149.

[14] B. Jungk, R. Petri, and M. Stöttinger, "Efficient Side-Channel Protections of ARX Ciphers," Cryptology ePrint Archive, Report 2018/693, 2018, https://eprint.iacr.org/2018/693.

[15] P. Schwabe and K. Stoffelen, "All the AES You Need on Cortex-M3 and M4," Cryptology ePrint Archive, Report 2016/714, 2016, https://eprint.iacr.org/2016/714.