The Phantom Gradient Attack: A Study of Replacement Functions for the XOR Function

Åvald Åslaugson Sommervoll(✉)

University of Oslo, Problemveien 7, 0315 Oslo, Norway
aavalds@ifi.uio.no
https://www.mn.uio.no/ifi/english/people/aca/aavalds/

Abstract. We build on the phantom gradient attack by introducing some new replacement function candidates for XOR. In this work, we put forward four new candidates' replacement functions and investigate the impact of different learning rates. We also extend and investigate the new replacement functions power on bitwise rotation XOR, of which previous phantom gradient attack works have struggled.

Keywords: Phantom gradient attack · Replacement function · Neural network · Neuro-cryptanalysis

1 Introduction

The recent publication in ICISSP 2021, *Dreaming of keys: introducing the phantom gradient attack* by Sommervoll [12], showed a new cryptanalytical approach. This work tries to unite the usually disjoint fields of algorithmic level of cryptanalysis with the heavily researched neural network training. The initial results for simple cryptographic functions were promising, but the attacks on the more complex XOR functions were not encouraging. In this paper, we present attacks on the XOR function using the *phantom gradient attack*.

As almost modern communication is done using bits, modern cryptographic functions typically work on a bitwise level. Moreover, cryptographic algorithms can be represented as a sequence of bitwise operations. That is, a symmetric key cryptographic encryption can be viewed as a function f can be broken down into multiple subfunctions $f_0, f_1, ..., f_n$ making an encryption:

$$f_{enc}(k, p) = f_0^\circ f_1^\circ ... ^\circ f_n(k, p) = c, \tag{1}$$

where k is the symmetric key, p is the plaintext and c is the resulting ciphertext. This disjoint processing of information can be viewed as different layers of a neural network. However, a challenge is that these f_i's are typically discrete operations that do not have any gradient. The *phantom gradient attack* therefore works by replacing them with piecewise continuous ones. By replacing the subfunctions with, *replacement functions* allow our neural network representation of the

X. Yuan et al. (Eds.): QShine 2021, LNICST 402, pp. 228–238, 2021.
https://doi.org/10.1007/978-3-030-91424-0_14

cryptographic algorithm to have *gradients*. Part of the challenge that Sommervoll [12] put forward in his paper was to find good such *replacement functions*, of which will be the focus of this paper. In Sommervoll's network, he assumed that the plaintext was fixed and tried to recover the unknown key k, we abstract away from such problems and focus exclusively on finding good replacement functions for the XOR function. Moreover, we aim to find a better replacement function than the one presented in Sommervoll's paper. An improved replacement function is a key ingredient to more successful phantom gradient attacks.

The remaining paper is organized as follows: Sect. 2 discusses related work. Next, Sect. 3 discusses replacement functions, their most important qualities, and which replacement functions we will be looking at in this paper. In Sect. 3.1, we analyze the replacement functions' performance on XOR between 3 inputs. Section 3.2 increases the complexity by analyzing their performance on XOR between three bitwise rotated inputs. A more complicated example, ASCONS Σ_1 permutation, is tested in Sect. 3.3. Finally, Sect. 4 concludes and provides a short security discussion.

2 Related Work

The phantom gradient attack introduced by Sommervoll in *Dreaming of keys: introducing the phantom gradient attack* [12], is a recent addition to the field of neuro-cryptology. A field that has seen limited growth since Dourlens introduced it in 1996 [4]. Especially the field of neuro-cryptography has had limited contributions since Kinzel and Kanter in 2002 introduced a neural cryptosystem [7], which was quickly broken by Klimov et al. [8] the same year. On the other hand, neuro-cryptanalysis has been catching some wind in recent years, with Alini successfully applying an attack on DES and Triple-DES using neuro-Cryptanalysis in 2012s [1], and So applying a deep learning-based attack on simplified DES, and round reduced Simon and Speck [11]. While the works by Alani, So, and Sommervoll are all instances of neuro-cryptanalysis, they all differ in their approach. All three works assume to be in the known plaintext case; in contrast to the others, Alani does not try to recover the key. Instead, he tries to simulate the decryption under an unknown key by feeding a neural network the ciphertext as input and assigning loss based on how close the output is to the expected plaintext. On the other hand, So's attack tries to train a deep network to guess the key by giving both the ciphertext and the plaintext as inputs and having the key as the desired output. By training the network like this, he uses this trained network to predict a possible key given the input-ed cipher- and plaintext pair. Sommervoll, with his phantom gradient attack, defines the network to be trained in that the known plaintext is integrated as part of the network's fixed weights, and the desired target is the ciphertext. While the input is initially a guessed key, which is "trained" and permuted in a similar manner to how adversarial examples are created for image recognition. A weakness to this approach is that since the gradients are only given by the replacement functions, there is the possibility of choosing bad phantom gradients, which may lead the attack astray. However, we may draw from the field of adversarial examples; Goodfellow et al.

found that “linear models lack the capacity to resist adversarial perturbation" in their work *Explaining and Harnessing Adversarial Examples* [5]. Thereby, if we have the freedom to choose linear functions as replacement functions, this may be favorable in our endeavor to find candidate keys. On the note of adversarial examples, some works only alter parts of an image like Su et al., which introduce adversarial examples that only alter one pixel [13] and Gritsenko et al. with briar patches that only affect a portion of the image [6]. This is especially interesting as some portions of cryptographic functions' initial state is known, like in the ASCON cryptosystem [3]. Therefore, a phantom gradient attack on such a cryptosystem should make sure not to alter the initial state's known parts.

3 Replacement Functions XOR

The most important quality for a replacement function is that the function and its discrete counterpart should have the same output given the same input. For the XOR function this means that inputs [1,1] and [0,0], should result in 0 and the inputs [1,0] and [0,1] should result in 1. This operation can be viewed as addition under modulo 2, which naturally gives us our first replacement function:

$$f(x, y) = x + y \ (mod\ 2). \tag{xori0}$$

We will call this replacement function *xori0*, as it is the most natural replacement function for *xor* between *i*ndexes. Furthermore, its derivatives are quite simple:

$$\frac{\partial f}{\partial x} = 1 \tag{2}$$

$$\frac{\partial f}{\partial y} = 1, \tag{3}$$

This representation is also linear, which is be favorable for generating adversarial examples [5]. Sommervoll also mentioned that XOR can be viewed as an addition under *mod*2 but did not consider it as a possible replacement function [12]. He did however consider what we will refer to as xori1:

$$f(x, y) = x + y - 2xy, \tag{xori1}$$

which had the unfavorable quality of having derivatives that are 0 for 0.5, midway between the bitshift from 0 and 1, namely:

$$\frac{\partial f}{\partial x} = 1 - 2y \tag{4}$$

$$\frac{\partial f}{\partial y} = 1 - 2x, \tag{5}$$

Our third candidate is a natural extension of xori1, without the weakness of having a fixed zero gradient between 0 and 1:

$$f(x, y) = (x - y)^2 = x^2 + y^2 - 2xy, \tag{xori2}$$

which has gradients that are 0 for x = y, which will be rare especially given a random initial guess. Our 4th replacement function *xori3* views the second index as a constant and splits the output into two separate cases, where the input x is either bitflipped or not depending on y:

$$f(x,y) = \begin{cases} x \text{ for } y \leq 0.5 \\ 1 - x \text{ for } y > 0.5 \end{cases} \tag{xori3}$$

This gives us our second linear replacement function, also making it especially vulnerable to adversarial examples [5]. Our 5th and final replacement function is xori4 which again is simple addition, but switches out the activation function mod2 which xori0 uses, and instead utilizes a sine-based activation function: $g(x) = \frac{1+sin(\pi x - \frac{\pi}{2})}{2}$, so we have:

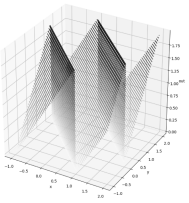
$$f(x,y) = \frac{1 + sin(\pi(x+y) - \frac{\pi}{2})}{2} \tag{xori4}$$

This variation of xori0 is differentiable everywhere, which is favorable from a mathematical perspective. However, in the context of neural networks, this property seems to hold little significance as many state-of-the-art activation functions are not differentiable everywhere, for example, ReLU [10]. To visualize these activation functions Fig. 1 shows how the different XOR functions behave in the interval from −1 to 2. All these replacement functions look quite different apart from all of them sharing the same final output for the binary inputs 1 and 0. Eq. (xori0) and Eq. (xori4) look similar since they both just use addition and some form of activation function to restrict the output. Similarly, Eq. (xori1) and Eq. (xori2) are similar as they are include the interaction term xy, and have no activation function. The odd one out in the group is definetly Eq. (xori3) as it takes a more discrete approach treating y as either a 1 in xor or a 0 in xor. These five xori functions differ in mathematical complexity, and there are no apriori reasons for the supremacy of one over the others. We use the same simplified model as used in Sommervoll's paper [12], Fig. 2: where FFNN stands for feed-forward neural network. For the network, we have four[1]. possible outputs and potentially infinitely many different starting values. We choose 1000 different starting values and try to recover all four of the different states from each of these 1000 different starting values. Table 1 shows the pct success rate of the different xor replacement functions for different learning rates, when run for 1000 generations.

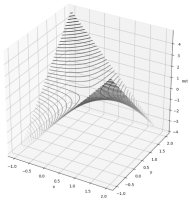
We see that all the replacement functions perform reasonably well for this simple example, especially with a learning rate of 0.2, where all of them get 100% recovery across all 4000 trials. Moreover, we see that a learning rate of 0.001 is a bit low for only 1000 iterations[2]. Aside from this, we see that both the linear replacement functions xori0 and xori3 perform extraordinarily well with

[1] The four possible 2 bit inputs are (0,0), (0,1), (1,0), and (1,1).

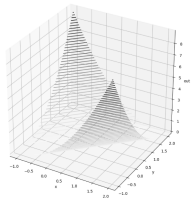
[2] For more intricate problems, we may need more iterations and perhaps an even lower learning rate.



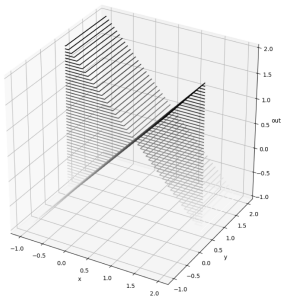
(a) eq. (xori0)



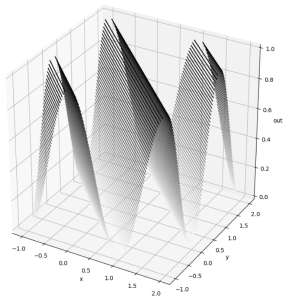
(b) eq. (xori1)



(c) eq. (xori2)



(d) eq. (xori3)



(e) eq. (xori4)

Fig. 1. View of the behaviour of the different XOR implementations in the range -1 to 2

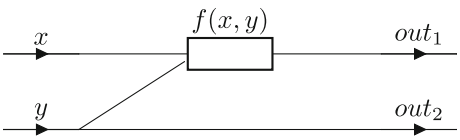


Fig. 2. Example FFNN for XOR between two inputs

Table 1. Percentage success rate of the the different XOR replacement functions across 1000 trials for each of the possible 2 bit outputs.

lr →	0.001	0.01	0.1	0.2	0.5	1.0
xori0	0.00	99.88	100.00	100.00	100.00	100.00
xori1	0.05	75.47	96.08	100.00	97.12	0.08
xori2	0.48	31.08	100.00	100.00	49.65	30.90
xori3	0.00	100.00	100.00	100.00	100.00	100.00
xori4	0.08	7.68	100.00	100.00	100.00	100.00

the higher learning rates with almost 100% recovery rate for every instance. Also, in contrast to what Parascandolo et al. [9] found, we see that the sine activation function used in *xori4* performs very well in this example, outperforming both xori1 and xori2, in all learning rates except for 0.01. Perhaps surprisingly, we also observe that Sommervoll's xori1 performs reasonably well with a learning rate between 0.1 and 0.5; however, we will see how this strength holds up as we increase the complexity of the problem.

3.1 XOR Between Three Inputs

Some cryptographic functions include a three-way XOR between indices; these indices can be complicated and, as is evident from Sommervoll's limited success with such functions [12]. We can illustrate this three-way XOR problem as a FFNN in the way shown in Fig. 3. This construction is straightforward and does

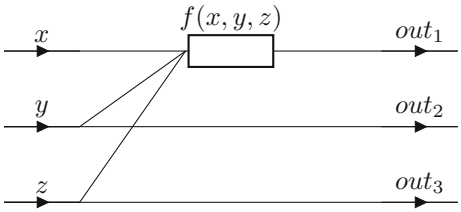


Fig. 3. Example FFNN for XOR between three inputs

not pose a much more significant challenge than XOR between two inputs. So we will not be looking at the phantom gradient attack on this network but instead, use it as an example to extend our pre-existing xori-functions, Eqs. (xori0) to (xori4). Extending xori0 is very simple we let:

$$f(x, y, z) = xori0(xori0(x, y), z) = x + y + z \ (mod\ 2), \tag{xorti0}$$

we call this xorti0, because it is the natural extension of xori0 and it is an *XOR* between *t*hree *i*nputs. For Eq. (xori1) we will use Sommervoll's [12] extension:

$$f(x, y, z) = x + y + z - 2xy - 2xz - 2yz + 4xyz, \tag{xorti1}$$

of which is the same as
$xori1(xori1(x, y), z) = xori1(xori1(x, z), y) = xori1(xori1(y, z), x)$. For xori2, on the other hand, it is a little bit more complicated. We have four natural candidates:

$$f(x, y, z) = x^2 + y^2 + z^2 - 2xy - 2xz - 2yz + 4xyz \tag{xorti2}$$

$$f(x,y,z) = f(f(x,y),z) = ((x-y)^2 - z)^2$$
$$= x^4 + y^4 + z^2 + 2x^2y^2 - 4x^3y - 4xy^3 + 4x^2y^2 - 2x^2z + y^2z - 2xyz \quad \text{(xorti2z)}$$

$$f(x,y,z) = f(f(x,z),y) = ((x-z)^2 - y)^2$$
$$= x^4 + z^4 + y^2 + 2x^2z^2 - 4x^3z - 4xz^3 + 4x^2z^2 - 2x^2y + z^2y - 2xzy \quad \text{(xorti2y)}$$

$$f(x,y,z) = f(f(y,z),x) = ((z-y)^2 - x)^2$$
$$= z^4 + y^4 + x^2 + 2z^2y^2 - 4z^3y - 4zy^3 + 4z^2y^2 - 2z^2x + y^2x - 2zyx, \quad \text{(xorti2x)}$$

where Eq. (xorti2) is based on Eq. (xorti1) and symmetric across the three inputs we are XOR-ing, while Eqs. (xorti2z) to (xorti2x) the natural extention of Eq. (xori2), but vary with respect two which index is XOR-ed last. For Eq. (xori3) we treated the second index as an external index from the start; we extend this by doing the same with the third index.

$$f(x,y,z) = \begin{cases} x \text{ for } (y \leq 0.5 \wedge z \leq 0.5) \vee (y > 0.5 \wedge z > 0.5) \\ 1 - x \text{ for } (y > 0.5 \wedge z \leq 0.5) \vee (y \leq 0.5 \wedge z > 0.5) \end{cases} \quad \text{(xorti3)}$$

Equation (xori4) we extend the same way we extended Eq. (xori0) as they are both based on addition and an activation function.

$$f(x,y,z) = \frac{1 + sin(\pi * (x + y + z) - \frac{\pi}{2})}{2} \quad \text{(xorti4)}$$

With these xorti functions in mind, let us move on to XOR between three bitwise rotated instances of the input.

3.2 XOR with Bitwise Rotation

To ensure no loss of information in this three way bitwise rotated XOR we construct a network with 4 inputs as shown in Fig. 4. where we have 4 inputs and xor between rotations 0, 1 and 2, in other words it defines an XOR on the form:

$$x_i \oplus x_{i+1(mod4)} \oplus x_{i+2(mod4)}.$$

This setup means that we have 16 different inputs, moreover, the recovered bit sequence will be unique if recovered. So if we want to check each xorti's performance 250 times per target, we get 4000 trials per xorti. Furthermore, as we are working with bitwise rotation the three replacement functions Eqs. (xorti2z) to (xorti2x) are equivalent so we only check eq. (xorti2z). The resulting performance is shown in Table 2. We see clearly that with XOR between three indices, the learning quickly becomes more complex. Note that the main xorti1 previously used by Sommervoll performs poorly and never has a success rate above 0.2. Of the proposed replacement functions, the clear winner among the candidates is

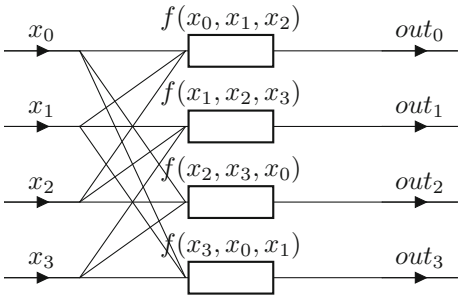


Fig. 4. XOR between three round rotated instances of a four-bit input

Table 2. Percentage success rate of the different XOR replacement functions on Fig. 4

lr →	0.01	0.1	0.2	0.5	1.0
xorti0	0.00	0.02	0.00	0.00	99.95
xorti1	16.98	17.10	16.58	9.93	0.00
xorti2z	4.70	29.88	17.00	7.80	2.92
xorti2	25.48	31.82	11.40	4.68	4.58
xorti3	21.38	19.85	18.60	100.00	100.00
xorti4	0.00	0.02	0.75	22.35	7.85

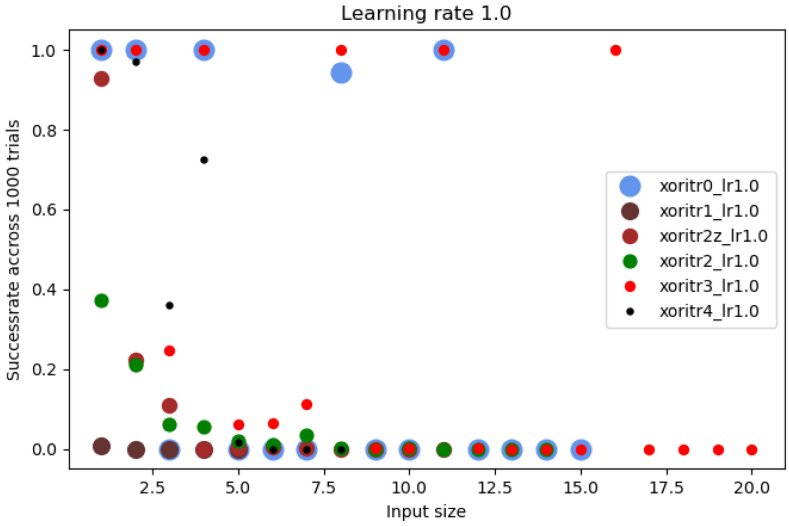
xorti3, which gets a 100% success rate for both learning rate 0.5 and learning rate 1. Also, note that xorti0 gets almost a 100% recovery rate for learning rate 1. It is quite surprising that such high learning rates are the ones that perform the best, which in contrast to the general case in neural network training. For example, for stochastic gradient descent in KERAS, the default value is 0.01 [2], it is even lower for some of the more fine-tuned optimizers. Also, perhaps surprisingly, we see that the xorti's based on addition and an activation function (XORITR0 and XORITR4) both seem to favor higher learning rates. In contrast, the continuous ones such as XORITR1, XORITR2, and XORITR2z seem to favor more midrange learning rates such as 0.1. Maybe with more iterations and better optimizers, they can perform even better.

3.3 ASCON's Σ_1 permutation

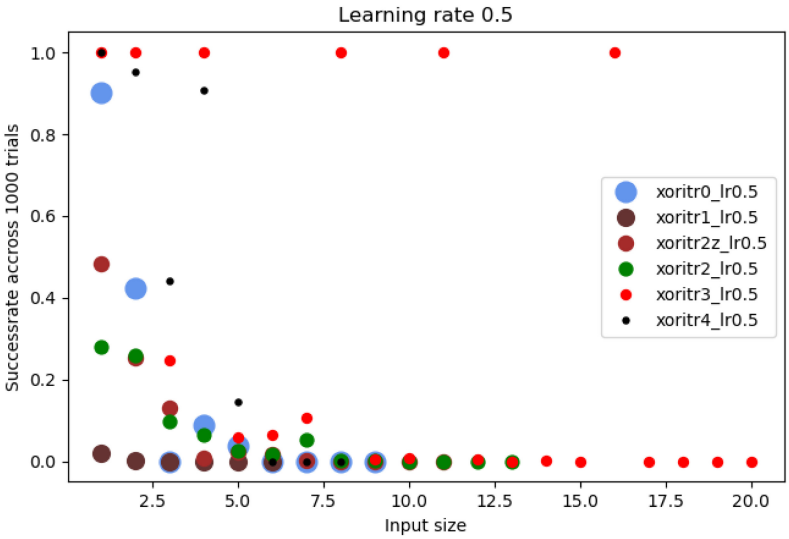
The cryptosystem ASCON has some instances of XOR between three bitwise rotated instances of inputs [3]. One of which is the Σ_1 permutation:

$$x_i \oplus x_{i+61(modn)} \oplus x_{i+39(modn)}, \tag{6}$$

where n is the input size, which in ASCON's case is 64. However, in our analysis, we will vary this input size to study our replacement functions' effectiveness. We



(a)



(b)

Fig. 5. Comparison of the different xorti's under the learning rates 0.5 and 1.0

wish to test input sizes from 1 up to 64, where input size four will be similar to the case we studied in Fig. 4 this time, it will be (i+1) and (i+3) instead. Similar to earlier trials, we run a 1000 iterations and a 1000 trials per input

size. However, we do not test all input sizes from 1 to 64; if all 1000 trials fail for four incrementally larger input sizes, we terminate the run and assume that it will also fail for larger block sizes. We do this for learning rates 1 and 0.5, and the results are shown in Fig. 5. We see that, like in Sommervoll's paper, they all perform rather poorly as we increase the number of bits. Sommervoll's earlier suggestions xoritr1 performs exceptionally bad, having no successes when dealing with more than 2 bits. Among the others, we see that xorti3 generally performs the best. This may be because of its semidiscrete nature. Also, it is influenced by fewer gradients simultaneously as y and z are treated as constants; however, this is not the entire story as xori0 performs similarly with a learning rate of 1.0. Some final tests with learning rates 0.2, 0.1 and 0.01, showed that xorti3 was successful in recovering the full 64 bits $\frac{25}{1000}$ trials with a learning rate of 0.2. This is still only a recovery rate of 2.5%. However, it shows that the phantom gradient attack can be successful on the full 64 bits.

4 Conclusion

In this work, we have put forward a series of candidate replacement functions for the XOR function. All of which performed well for a simple XOR between two indices. However, in the more complex case of XOR between three bitwise rotated instances of the input, the replacement functions perform considerably worse. Perhaps most interesting was that a considerably high learning rate was the best performing and that the more simplistic replacement functions also performed best. The piecewise differentiable xori3 and xorti3 performed the best, clearly outperforming Sommervoll's previous xori1 and xorti1. We also found some merit in attempting to use linear representations as it is easier to find adversarial examples in these cases.

The phantom gradient attack introduced by Sommervoll in 2021 does not yet pose any threat to state-of-the-art cryptosystems. The phantom gradient attack is heavily based on the training of neural networks, of which current state-of-the-art works best with deep networks, so any cryptosystem that employs a particularly wide network should be more robust. In this paper, we did show that we could recover 64 bits of a permutation 2.5% of the time. This is not enough to threaten most modern cryptosystems yet but can provide a building block for future attacks.

We iteratively run 1000 trials per xorti on the different input sizes 1 through 64 for the permutation shown in eq. (6). If the success rate is 0% for four input sizes in a row, then the run terminates, and we assume the larger input sizes also to achieve roughly 0% success.

Acknowledgement. The author wishes to give special thanks to Audun Jøsang and Thomas Gregersen for valuable discussion and words of encouragement.

References

1. Alani, M.M.: Neuro-cryptanalysis of DES and triple-DES. In: Huang, T., Zeng, Z., Li, C., Leung, C.S. (eds.) ICONIP 2012. LNCS, vol. 7667, pp. 637–646. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34500-5_75
2. Chollet, F.: Keras (2015).https://github.com/fchollet/keras
3. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2. submission to round 3 of the CAESAR competition (2016). https://competitions.cr.yp.to/round3/asconv12.pdf
4. Dourlens, S.: Applied neuro-cryptography and neuro-cryptanalysis of des. Master Thesis (1996). https://doi.org/10.13140/RG.2.2.35476.24960
5. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
6. Gritsenko, A.A., D'Amour, A., Atwood, J., Halpern, Y., Sculley, D.: Briarpatches: Pixel-space interventions for inducing demographic parity. arXiv preprint arXiv:1812.06869 (2018)
7. Kinzel, W., Kanter, I.: Neural cryptography. In: Proceedings of the 9th International Conference on Neural Information Processing 2002, ICONIP 2002, vol. 3, pp. 1351–1354. IEEE (2002)
8. Klimov, A., Mityagin, A., Shamir, A.: Analysis of neural cryptography. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 288–298. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36178-2_18
9. Parascandolo, G., Huttunen, H., Virtanen, T.: Taming the waves: sine as activation function in deep neural networks (2016)
10. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions. arXiv preprint arXiv:1710.05941 (2017)
11. So, J.: Deep learning-based cryptanalysis of lightweight block ciphers. Secur. Commun. Netw. **2020** (2020)
12. Sommervoll, Å.: Dreaming of keys: Introducing the phantom gradient attack. In: 7th International Conference on Information Systems Security and Privacy, ICISSP 2021, 11 February 2021 through 13 February 2021, SciTePress (2021)
13. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. IEEE Trans. Evol. Comput. **23**(5), 828–841 (2019)