Deep Neural Cryptography

Adi Shamir

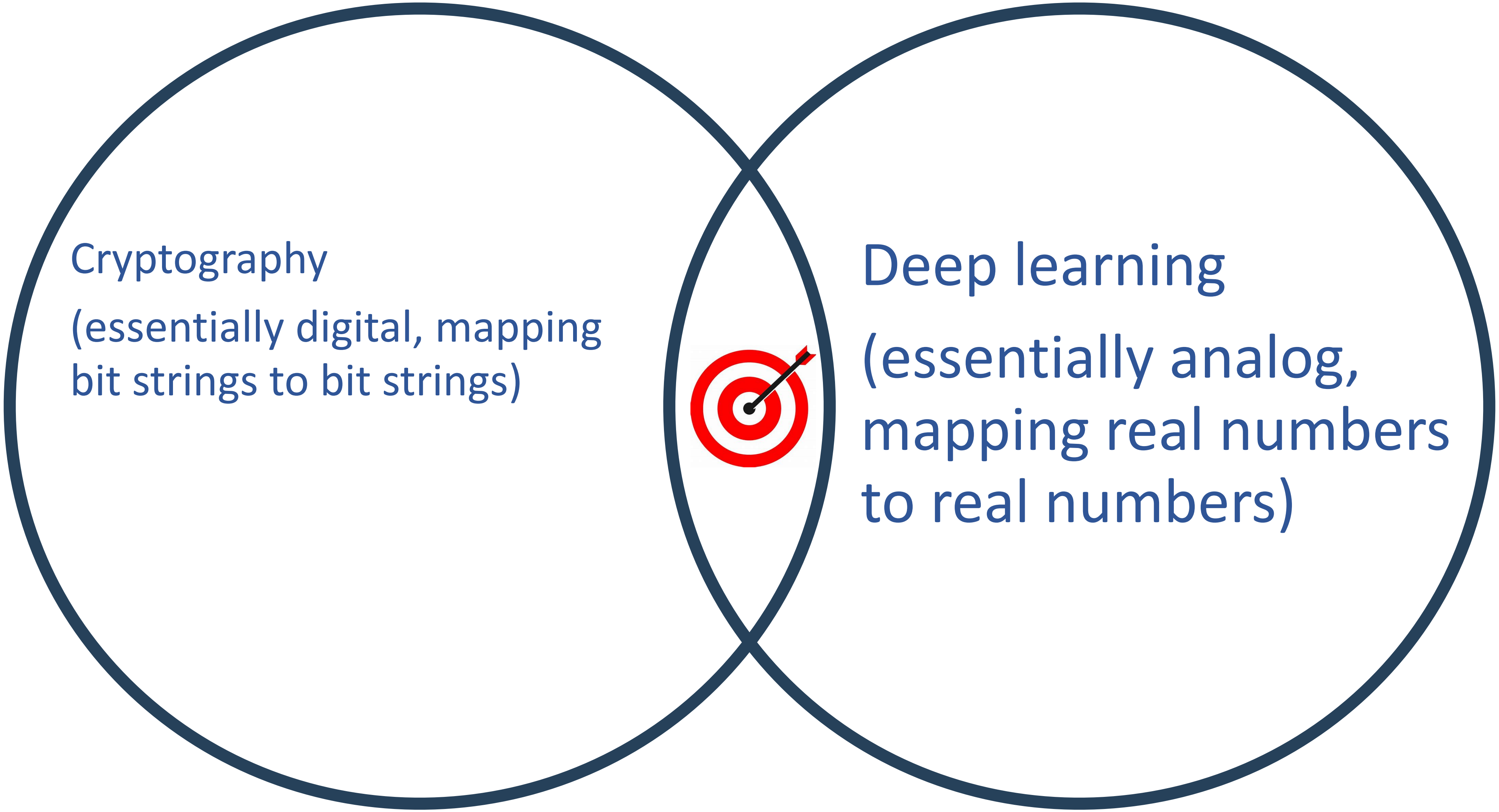
Computer Science Dept

Weizmann Institute of Science

Israel

Joint work with David Gerault (TII), Anna Hambitzer (TII), and
Eyal Ronen (Tel Aviv University)

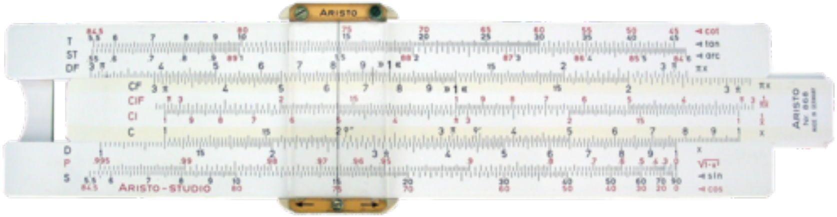
There are two major research areas in CS with almost no intersection so far:



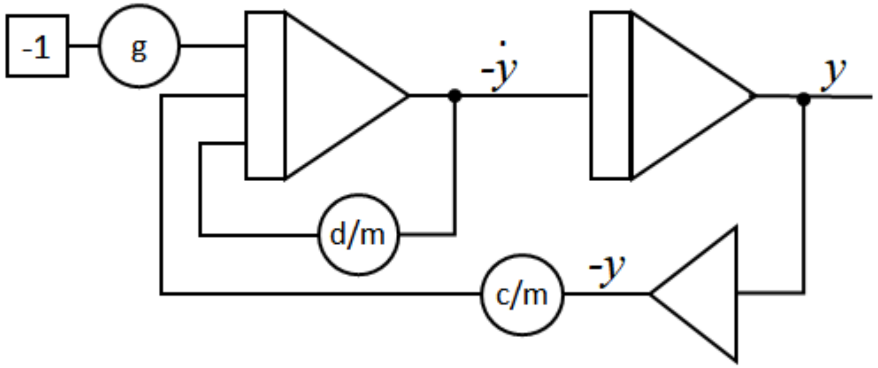
For 2000 years, we used analog computers

Antikythera mechanism

Solving differential equations



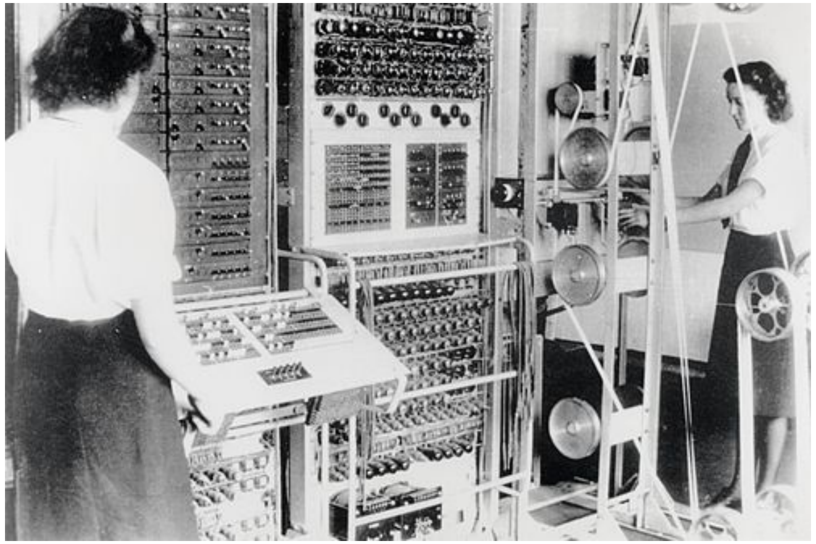
$$m\ddot{y} + d\dot{y} + cy = mg$$



The Norden bomb sight

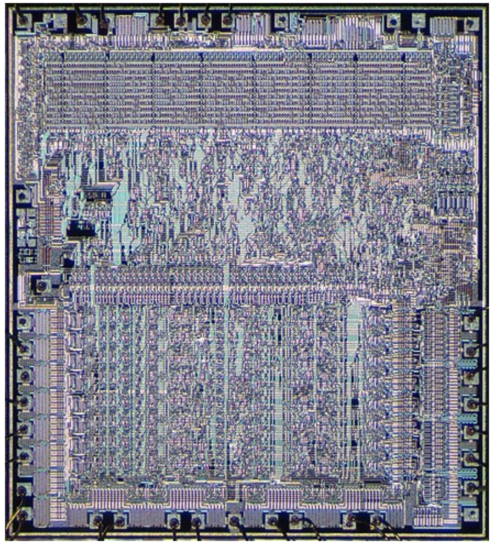
About 70 years ago, we completely switched to digital computers

The Colossus code breaking computer



IBM System 360

Microprocessors



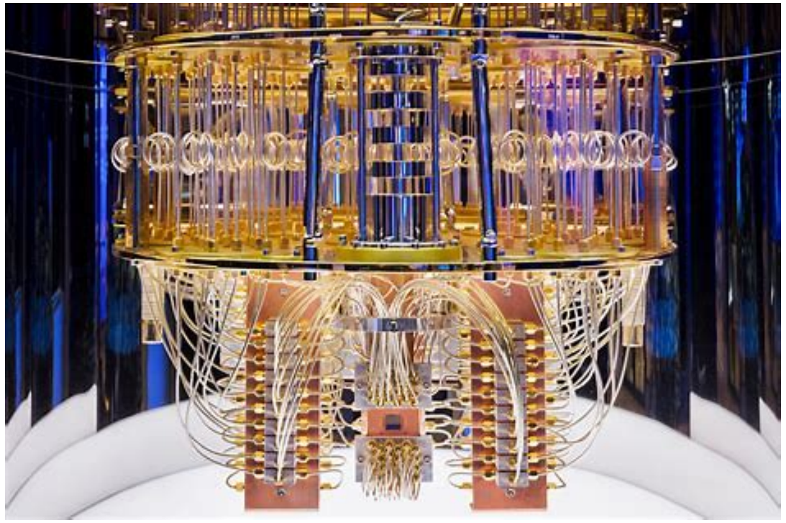
They are back! The MAGA movement of the last 20 years (Make Analog Great Again)

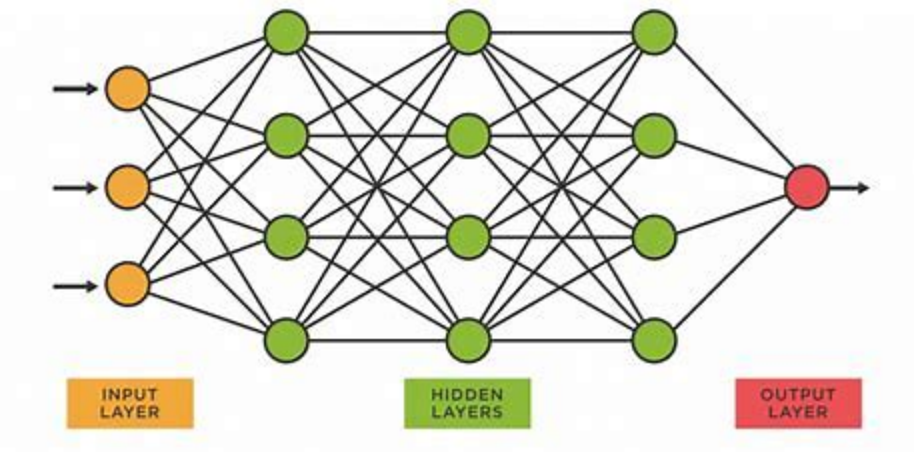
Quantum computers

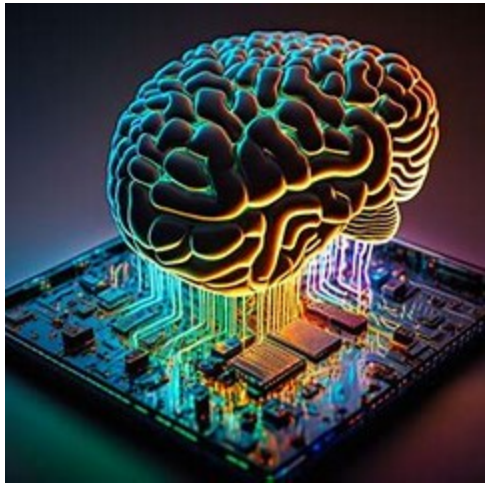
(with complex valued superpositions in qubits)

Deep Neural Networks (with real valued inputs and weights)

Neuromorphic computers (with time-coded spiking networks)

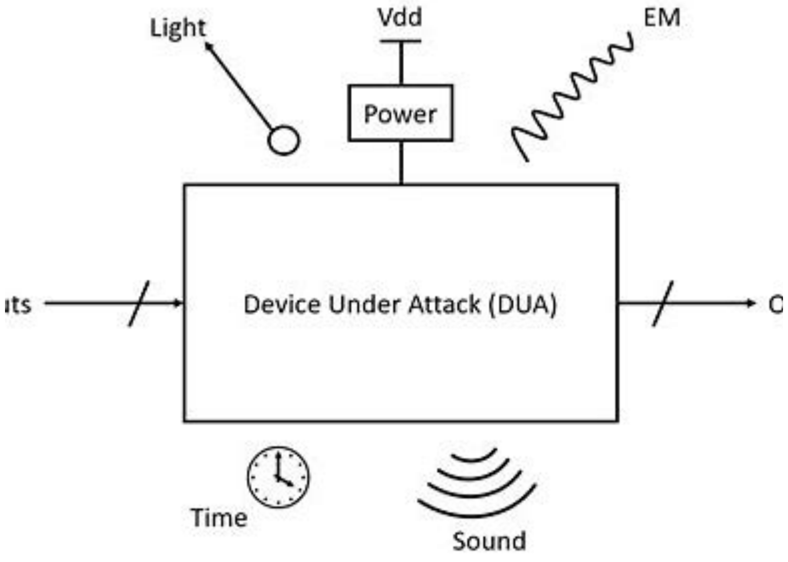




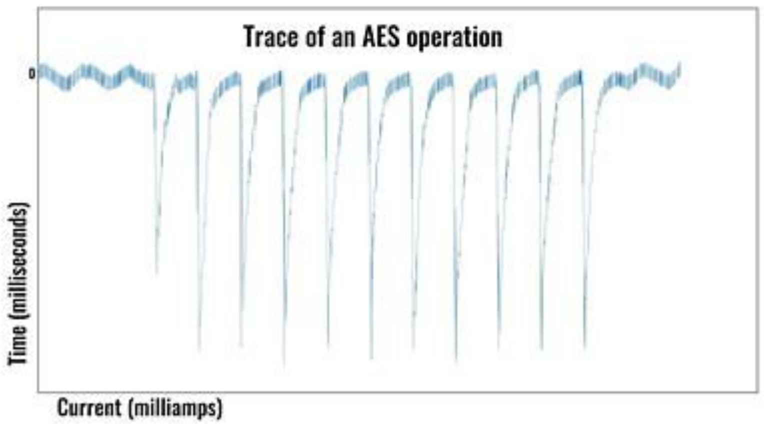


Even our digital computers have analog characteristics, used in side channel attacks

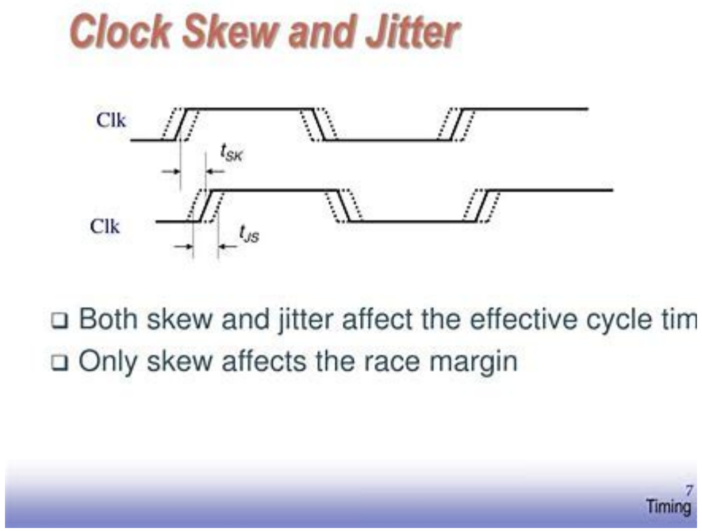
Various emanations from digital computers



Cryptanalysis of AES by analyzing power traces



Fault attacks with modified clock pulses



The landscape of analog/digital security

Users want to encrypt

Analog data (problematic)	digital data, digital computer	digital data, leaky computer	digital data, analog computer

The landscape of analog/digital security

users

adversaries		digital data, digital computer	digital data, leaky computer	digital data, analog computer
	digital attack			
	analog attack			

The landscape of analog/digital security

users

adversaries	digital data, digital computer	digital data, leaky computer	digital data, analog computer
digital attack	standard cryptography		
analog attack			

The landscape of analog/digital security

users

		digital data, digital computer	digital data, leaky computer	digital data, analog computer
adversaries	digital attack	standard cryptography		
	analog attack		side channel attacks	

The landscape of analog/digital security

users

adversaries

	digital data, digital computer	digital data, leaky computer	digital data, analog computer
digital attack	standard cryptography	unreasonable restriction	
analog attack		side channel attacks	

The landscape of analog/digital security

users

adversaries	digital data, digital computer	digital data, leaky computer	digital data, analog computer
digital attack	standard cryptography	unreasonable restriction	
analog attack	meaningless	side channel attacks	

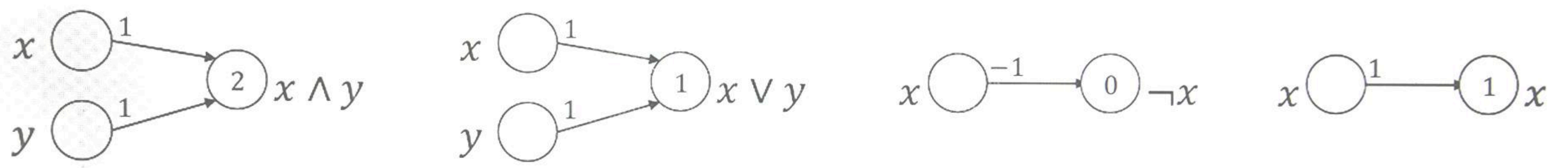
The Goldwasser, Kim, Vaikuntanathan, and Zamir FOCS'2022 Paper

- They considered the case in which the only allowed inputs to the DNN are zeroes and ones, and just used the universality of DNN's
- While technically correct, they missed all the fun…

B Universality of Neural Networks

A useful and seemingly essential property of good families of activation functions is their universality, i.e., the ability to represent every function using a neural network with activation functions from the family. For example, it is well-known that neural networks with perceptrons as their activation function (also called multi-layer perceptrons or MLPs) can realize any Boolean function.

Lemma B.1. *A single layer perceptron can realize boolean* AND, OR, NOT, *and* Repeat *gates.*



The landscape of analog/digital security

users

adversaries	digital data, digital computer	digital data, leaky computer	digital data, analog computer
digital attack	standard cryptography	unreasonable restriction	Goldwasser+ FOCS 2022
analog attack	meaningless	side channel attacks	

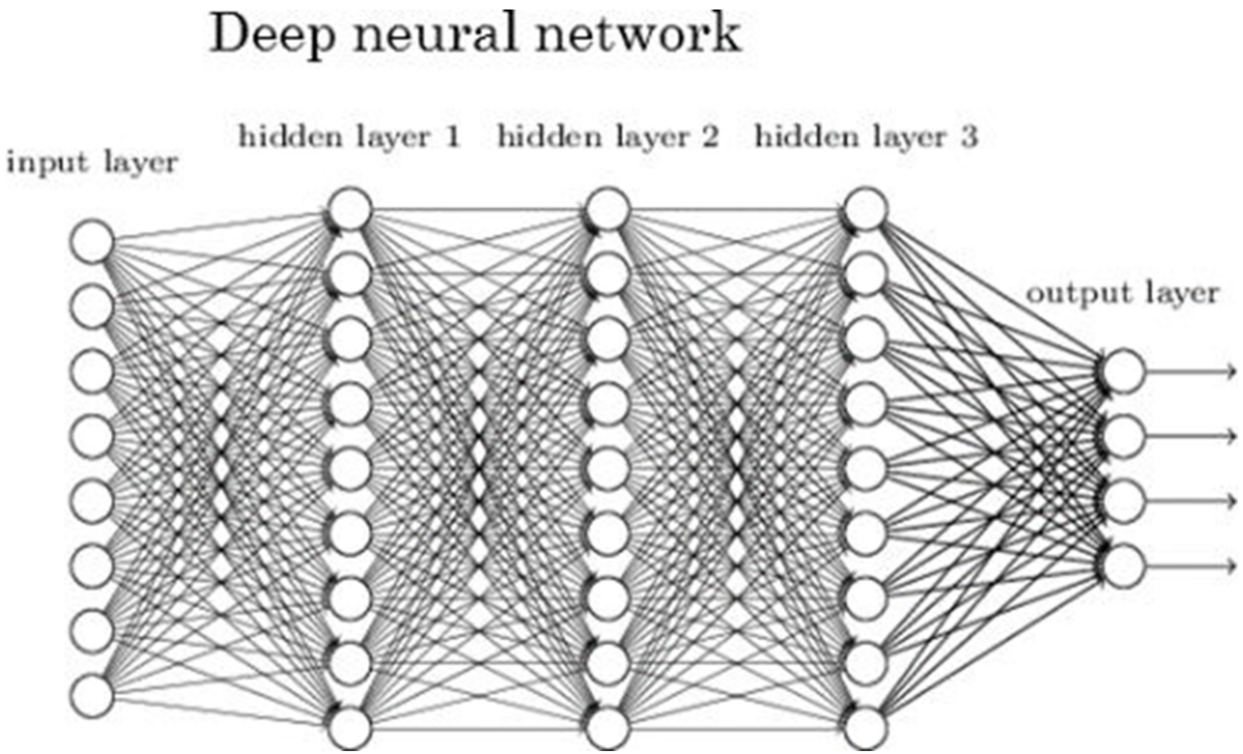
The landscape of analog/digital security

users

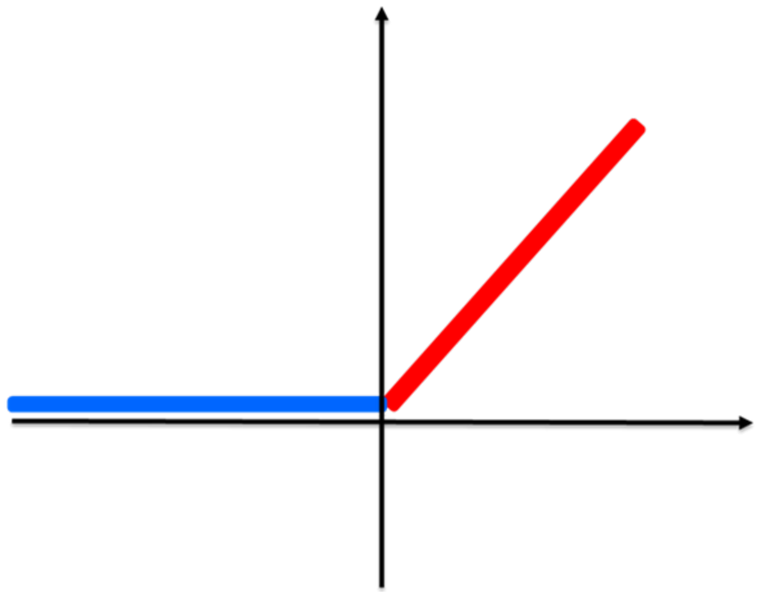
adversaries	digital data, digital computer	digital data, leaky computer	digital data, analog computer
digital attack	standard cryptography	unreasonable restriction	Goldwasser+ FOCS 2022
analog attack	meaningless	side channel attacks	this paper

The layered structure of DNN computers

- Deep neural networks have multiple layers, where each layer typically consists of a linear mapping with real valued coefficients followed by the ReLU activation functions applied to all its outputs



ReLU(x)=MAX(x,0)



Can we implement digital cryptography on such an analog computational model?

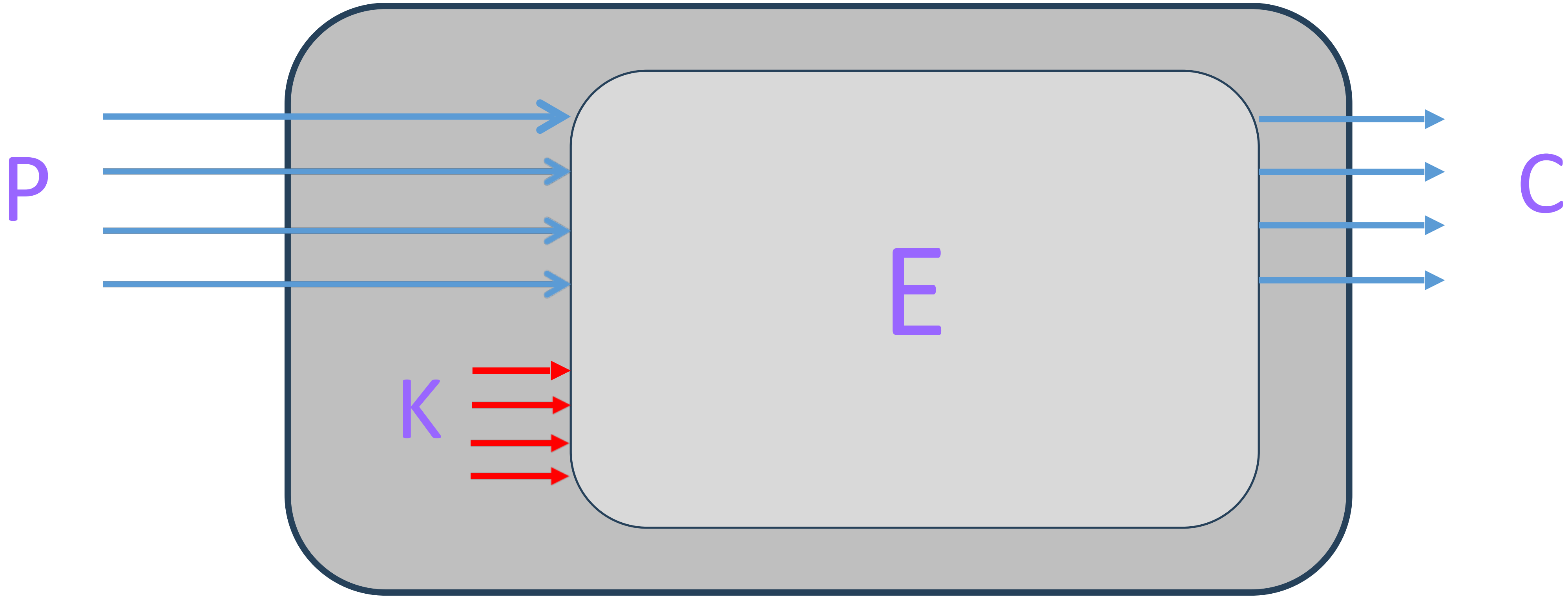
- ReLU-based DNN's are digitally universal, since they can easily implement the basic Boolean functions of "AND" "OR" and "NOT"

- ReLU-based DNN's are also analogly universal, since they can approximate any continuous real-valued function using a sufficiently wide network with one hidden layer

- While we already know that DNN's can correctly implement any digital cryptographic function, the question whether such implementations are secure had not been analyzed so far

How secret keys are handled in the DNN

- The secret key can be provided as a sequence of additional inputs; we assume that the adversary cannot see or change these inputs



This is a totally new security playground with new rules and new techniques

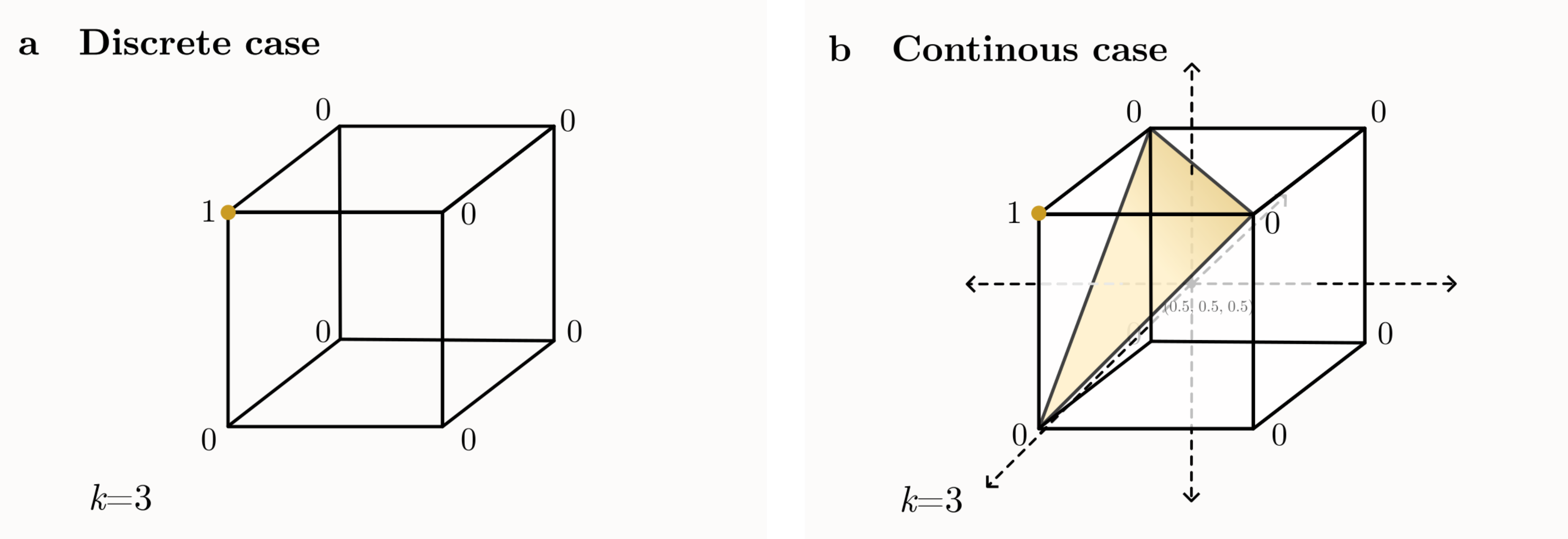
- For example, the attacker can ask the DNN to encrypt the “plaintext” P whose “bits” are (0.3, -7, Π, …) and obtain the “ciphertext” (-2.7, √2,…)

- For example, the attacker can apply a jitter attack, in which he increases or decreases the value of one “bit” by ±ε, and observe whether the “ciphertext” changes or not. This is a stronger form of differential cryptanalysis, where the attacker can only flip 0/1 values

- For example, the attacker can analyze the piecewise linear output as the input follows a straight line between two plaintexts P1 and P2, as I did in a Eurocrypt’24 paper on extracting the parameters of DNN’s

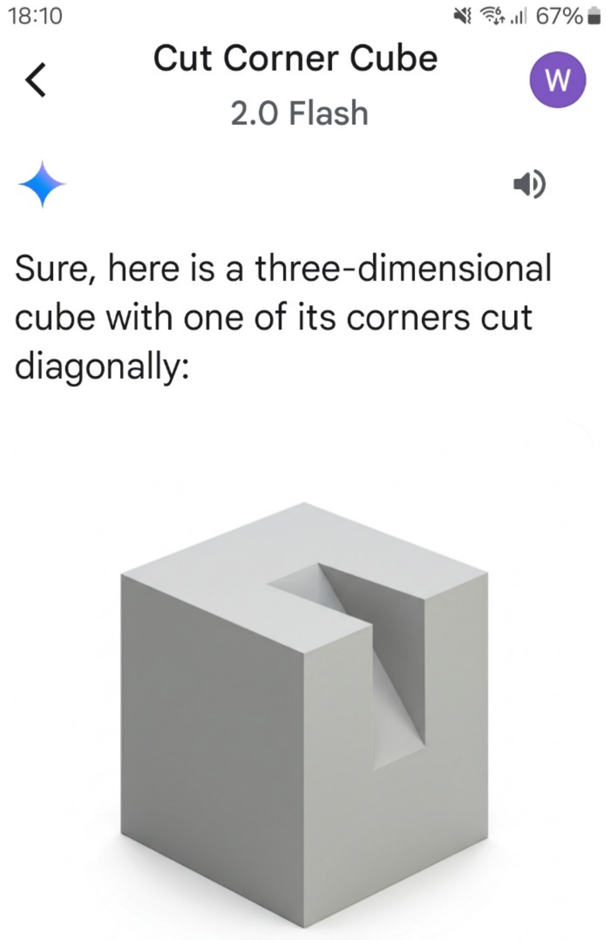
Our basic tool: corner functions

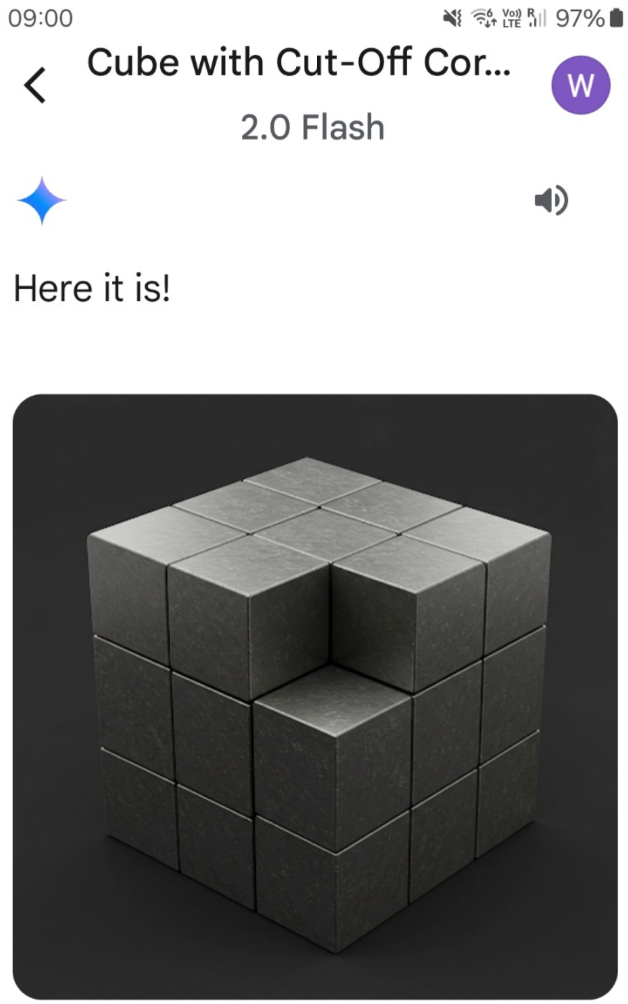
- Consider an n-dimensional Boolean cube with a single 1 at one of its corners, and 0's at all the other corners. We can easily implement this function with a single ReLU applied to a linear combination of its inputs
- For example, consider a 3D cube with a single 1 output at corner 101. This discrete mapping can be implemented with the continuous corner function ReLU(x1-x2+x3-1) which is 0 at all corners except 101

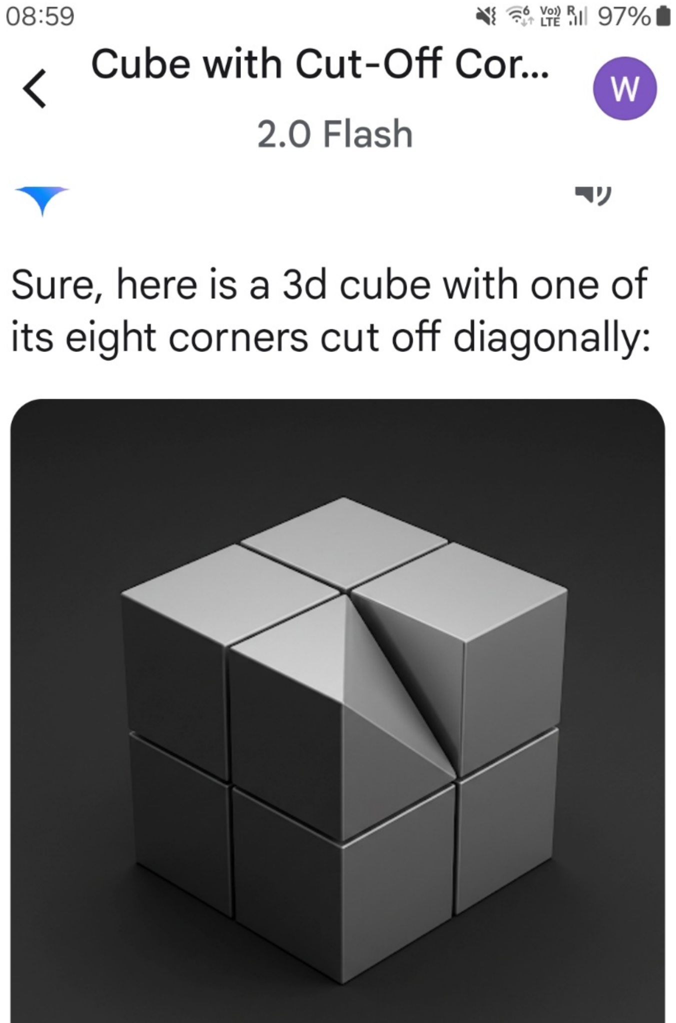


A side note: How intelligent are current LLM’s?

- To get a professional looking drawing of a corner function, I asked several leading generative AI systems to help me. Here is Gemini:







The difficulty of finding the special corner

- We now consider the following search problem: We are given a black box which implements some unknown corner function. How many queries are needed to find its special corner?

- If we are only allowed to query the black box with binary inputs, we need $\Omega(2^n)$ queries since we learn nothing from all the 0 answers

- (A side remark: This is exactly the search problem for which Grover's algorithm can improve the search complexity to $O(2^{n/2})$ when we allow superpositions of 0's and 1's on a quantum computer)

The difficulty of finding the special corner

- However, if we are allowed to query the black box with real valued inputs, we can find the special corner with just n queries by starting at the center of the cube (0.5, 0.5, …, 0.5), and move a distance n in any one of n main directions (to (0.5+n, 0.5, …, 0.5), …, (0.5, 0.5, …, 0.5+n)). A positive output produced for the i-th such query proves that this bit is 1 in the special corner, while a zero output proves that this bit is 0 in the special corner.

- This demonstrates a provably exponential gap between the query complexities of the search problem in the two computational models

How to implement Cryptography on DNN’s: The example of AES

- The way we usually construct DNN’s is via training
- We can collect a large number of plaintext/ciphertext pairs, and try to iteratively use gradient descent to optimize the weights of the network
- This had been tried multiple times, but always failed for AES, since a good cryptosystem destroys all the simple patterns in the training data
- The resultant network can easily memorize all the training examples, but can’t generalize the mapping to new inputs

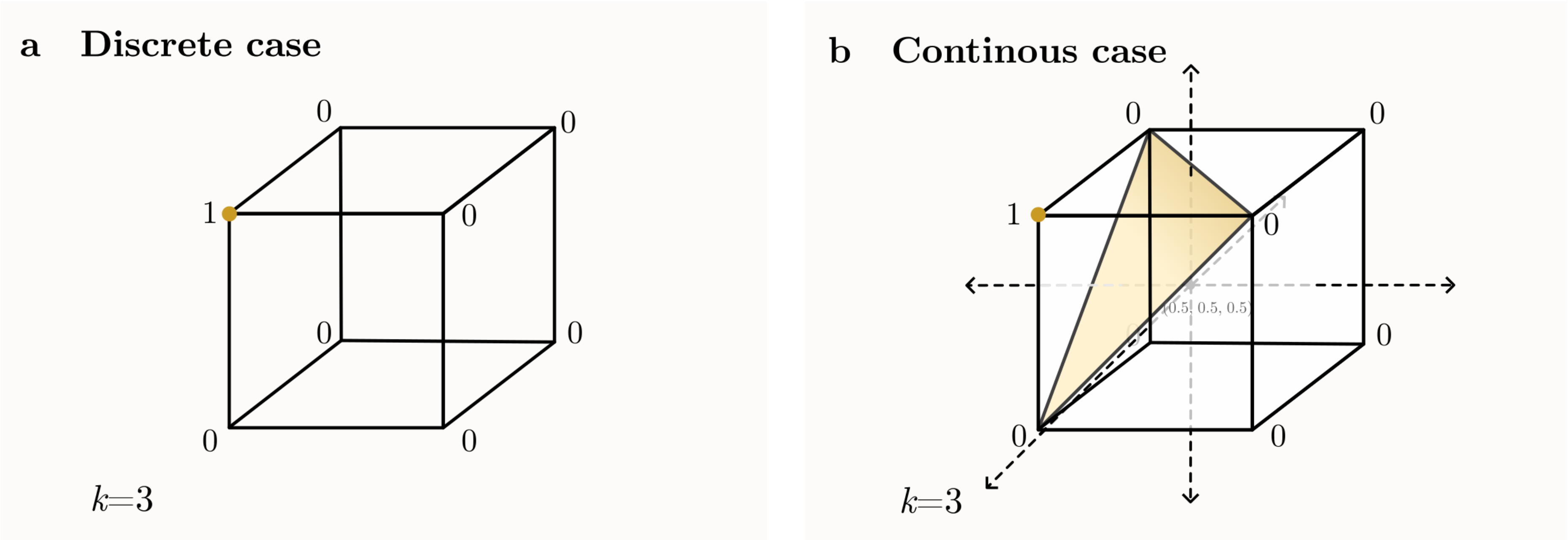
How to implement Cryptography on DNN's: The example of AES

- Everything in AES can be implemented with just two types of operations: Mapping 8-bit inputs to 8-bit outputs (Sbox, multiplication of a byte by the constants 2 and 3 in the AES finite field), and mapping 2-bit inputs to 1-bit outputs (XOR's of subkeys, and XOR's in the linear mixing)

- By using corner functions, we can implement any Boolean function with a small number of input bits as a simple ReLU-based DNN

Implementing Sbox using corner functions

- Consider the 8-dim Boolean cube which specifies one of the 8 output bits of the Sbox. Since the Sbox is balanced, exactly 128 of its 256 corners are labeled with 1 and the other 128 corners are labeled with 0
- For each one of the 128 corners labeled with 1, prepare a single neuron implementation of its corner function



Implementing Sbox using corner functions

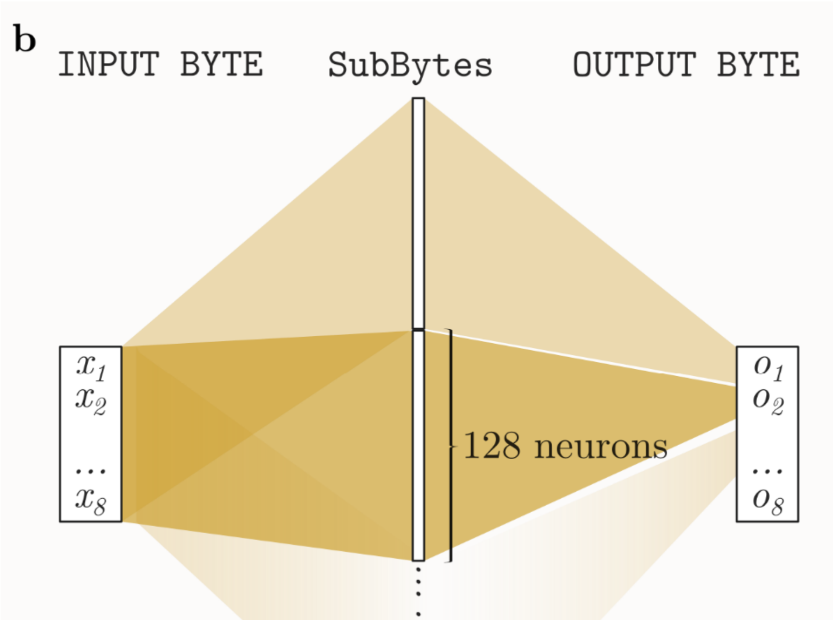
- Each output bit of the Sbox can be implemented as the sum of these 128 corner functions. This requires one hidden layer with 128 neurons

- To implement all 8 output bits, place 8x128=1024 neurons in parallel in the same hidden layer, and sum each group of 128 separately

a

INPUT BYTE — OUTPUT BYTE

x_1	x_2	x_3	x_4	...	x_8	o_1	o_2	...	o_8
0	0	0	0	...	0	1	1	...	0
1	0	0	0	...	0	1	1	...	0
1	1	0	0	...	0	1	1	...	0
.	.	.	.	.	.	.	.	.	.
1	1	1	1	...	1	0	1	...	0



To implement each XOR, use 2 neurons

- The definition XOR(x1,x2) = ReLU(x1-x2) + ReLU(x2-x1) is a special case of the general Boolean cube construction since XOR(x1,x2) is a 2-dim cube with two corners outputing 0 and two corners outputing 1

- We can thus implement everything in AES as sums of corner functions

- We call this the natural implementation of AES in a DNN

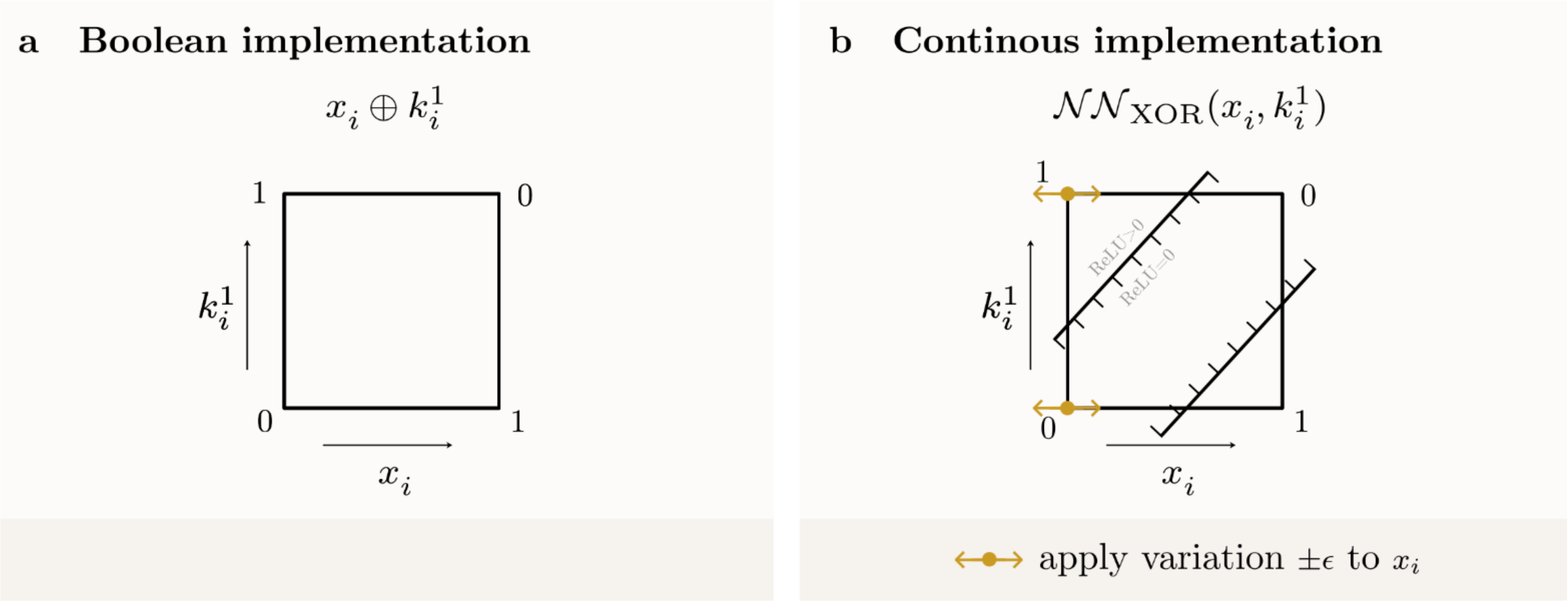
- It is correct in the sense that it computes the correct 0/1 outputs for any collection of 0/1 inputs; it computes something weird otherwise

Is this natural implementation secure when the adversary can use real valued inputs?

- The answer is that these implementations can be easily broken
- Almost any secret key block cipher (including AES) starts by XOR'ing each input bit xi with some key bit ki
- We will now show how to recover all the ki bits used in the first round of the encryption via a simple jitter attack

Attacking the natural implementation of XOR

- When implemented XOR as the sum of two separated corner functions:
- We can use xi=0, and distinguish between ki=0 and ki=1 by jittering xi: If ki=0 the output is always stable, while if ki=1 the output (usually) jitters

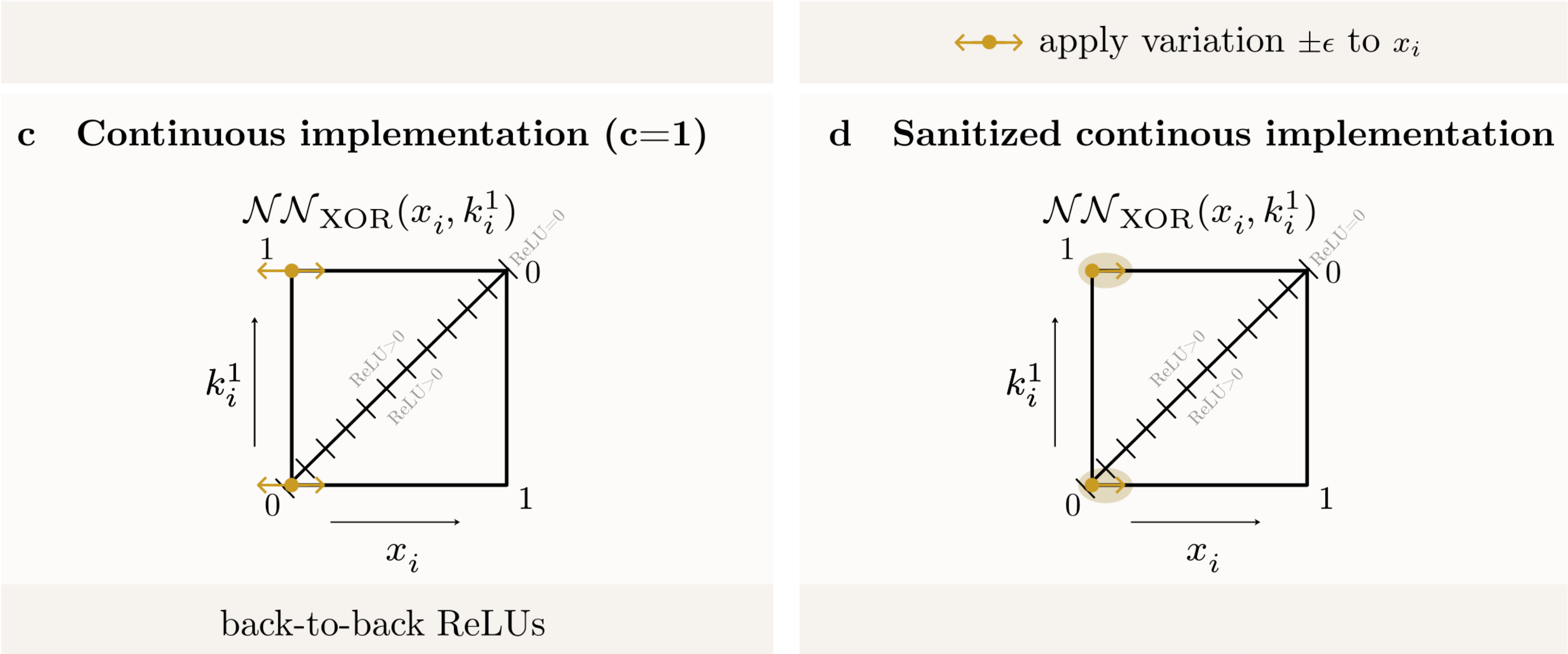


The natural implementation of bitwise XOR

- Implementing XOR as the sum of two back-to-back corner functions:

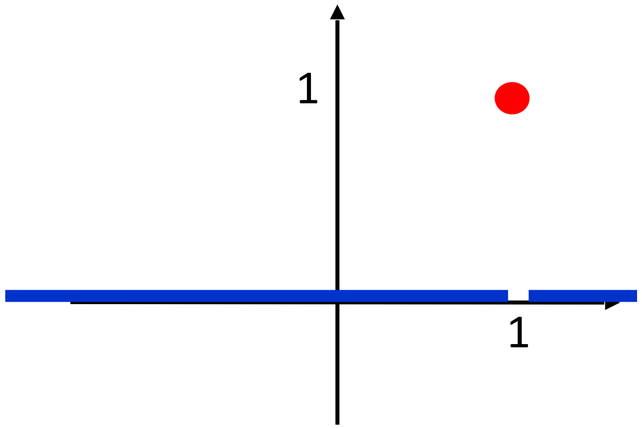
XOR(xi,ki) = ReLU(xi-ki) + ReLU(ki-xi)

- In case (c) we look for jitter symmetry; in the input-sanitized case (d) we can't jitter in both directions so we need a different kind of attack

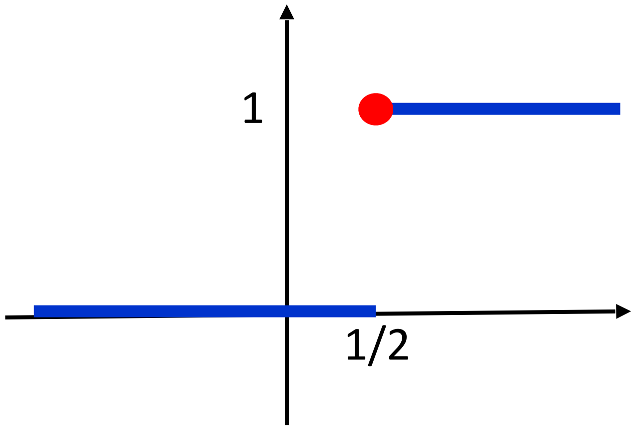


A potential solution: try to sanitize all input values to restrict the power of the adversary

Perfect sanitization, can't be implemented with ReLU's



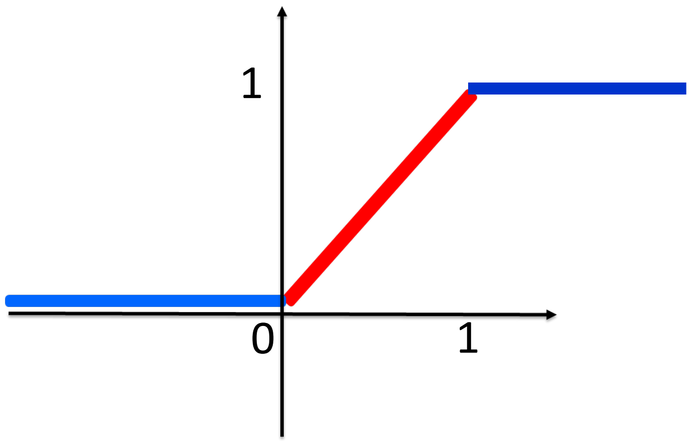
Perfect sanitization, can't be implemented with ReLU's



Partial sanitization can be realized with a simple DNN:

STEP(x)=ReLU(x)-ReLU(x-1)

Attacker can only use input values in the range [0,1]



Attacking the input-sanitized version of AES

- In AES, after XOR'ing a group of 8 input bits x1…x8 with 8 key bits k1,…k8, we map the resultant 8 bits y1,…y8 to z1…z8 via an 8-bit to 8-bit Sbox (i.e., z1…z8=Sbox(x1…x8 XOR k1…k8))

- Assume that each output bit zi of the Sbox is naturally implemented as a sum of 128 corner functions over the 8-dimensional cube of yi values

Attacking the input-sanitized version of AES

- When we jitter the input y1,…,y8 around any combination of 0/1 values, an output bit zi remains stable if and only if zi=0 for that input

- When we concatenate the 8 output bits z1,…z8, all of them remain stable simultaneously if and only if the 0/1 output of the Sbox is 0…0

- If at least one of the eight 0/1 outputs of the Sbox is not 0, the 8 output values of the Sbox will jitter, and this jitter is likely to avalanche all the way to the ciphertext values, which will also jitter

Attacking the input-sanitized version of AES

- We now have a way to test if the output of any particular Sbox in the first round of AES is z1,…,z8=00000000 ; this happens if and only if the input to this Sbox is y1,…,y8=01010010 . Since we know the plaintext bits x1…x8, we can now recover the 8 corresponding key bits as k1…k8 = x1…x8 XOR 01010010

- Repeating for all the 16 Sboxes in the first round of AES recovers the full 128 bit key

- This attack was experimentally verified using negligible time with 100% success rate

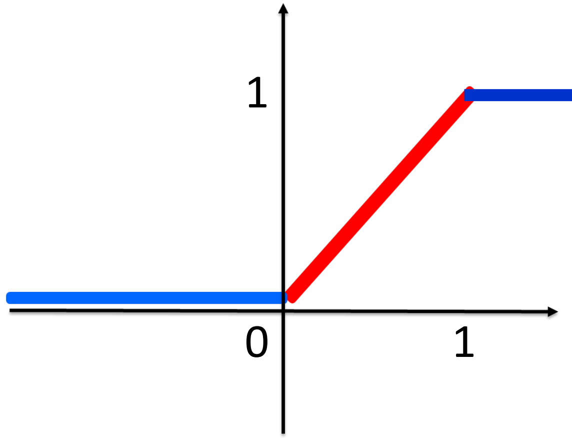
Can we find a different implementation of AES which is secure against any such attack?

- At first we were skeptical, since attackers have so much additional power in this analog model of computation (as in the case of side channel attacks, where no perfectly secure solutions are known)

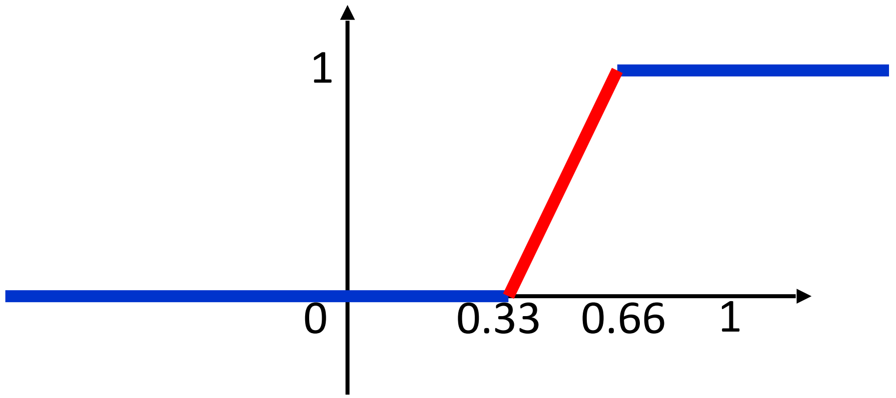
- However, after thinking hard, we found a provably secure way to implement any cryptographic functionality in a ReLU-based DNN

First step: sanitize the inputs more tightly

- Apply a tighter step function to each input separately:



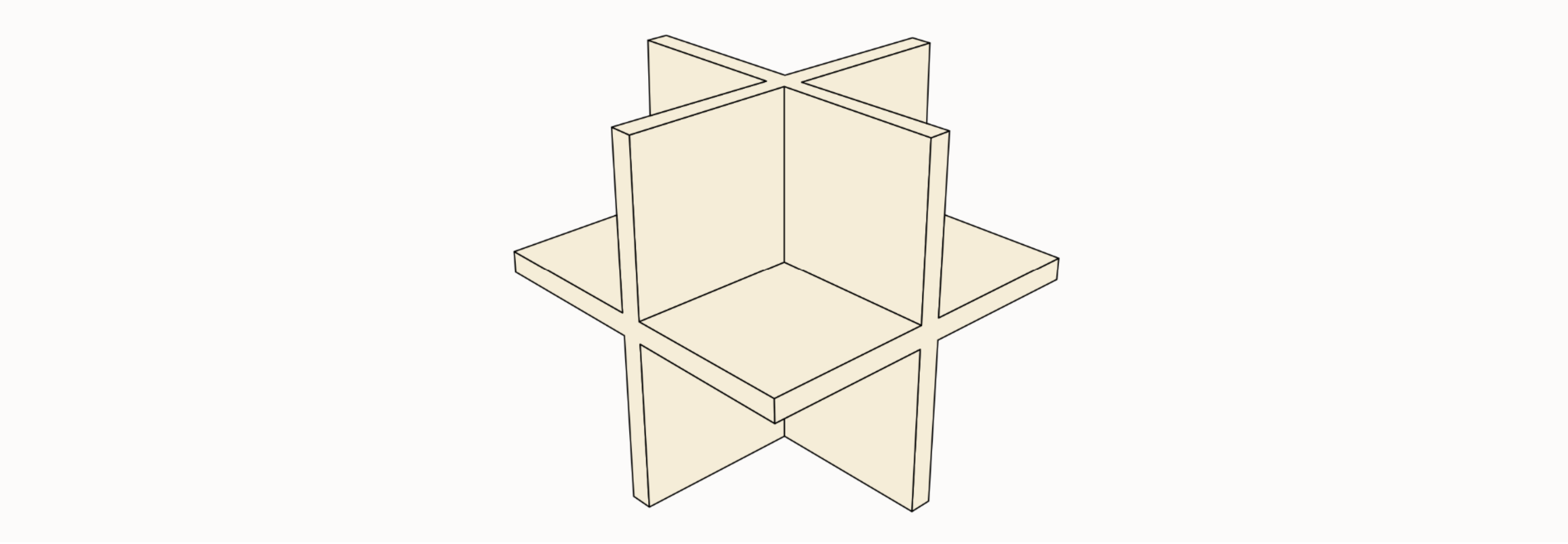
OLD-STEP(x)=ReLU(x)-ReLU(x-1)



STEP(x)=3*(ReLU(x-0.33)-ReLU(x-0.66))

Second step: Identify the "danger zone"

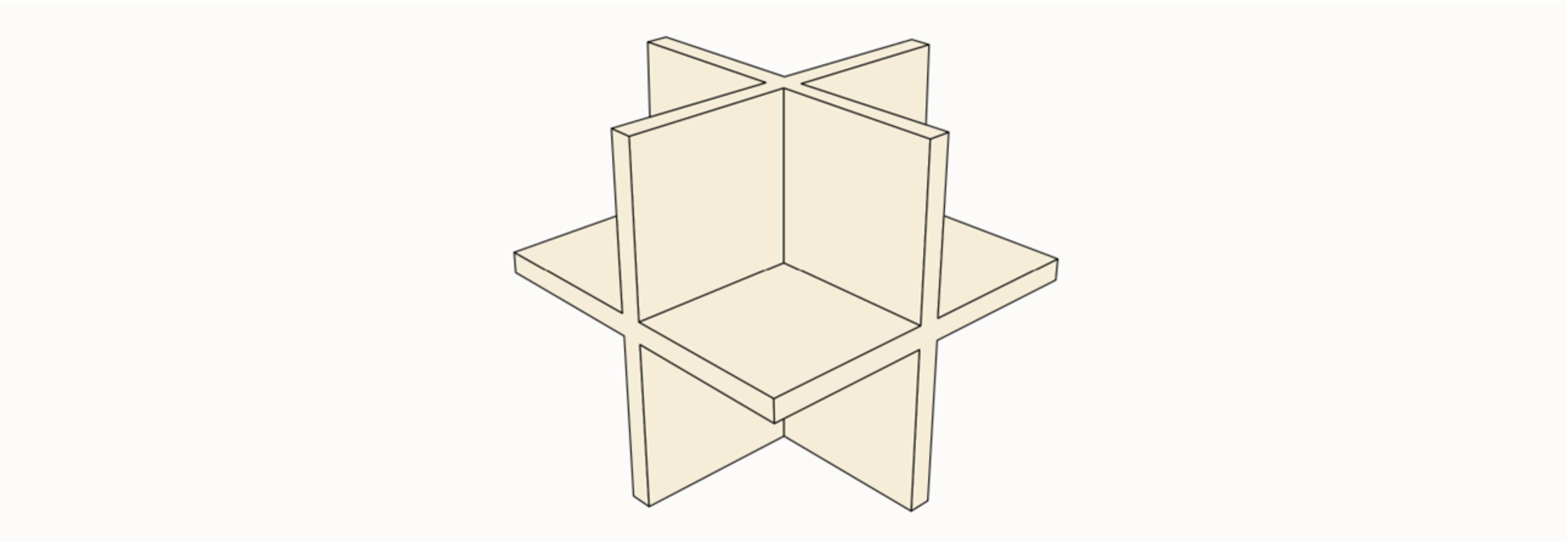
- Consider the multiwall in the input space, which is the high dimensional cross where at least one input coordinate lies between 0.33 to 0.66
- This is the "danger zone" where the sanitized inputs may not be 0 or 1
- In each orthant (separated from all other orthants by the multiwall), the sanitized inputs are a constant binary strings of just 0's and 1's



Third step: force all outputs for inputs in the "danger zone" to be identically zero

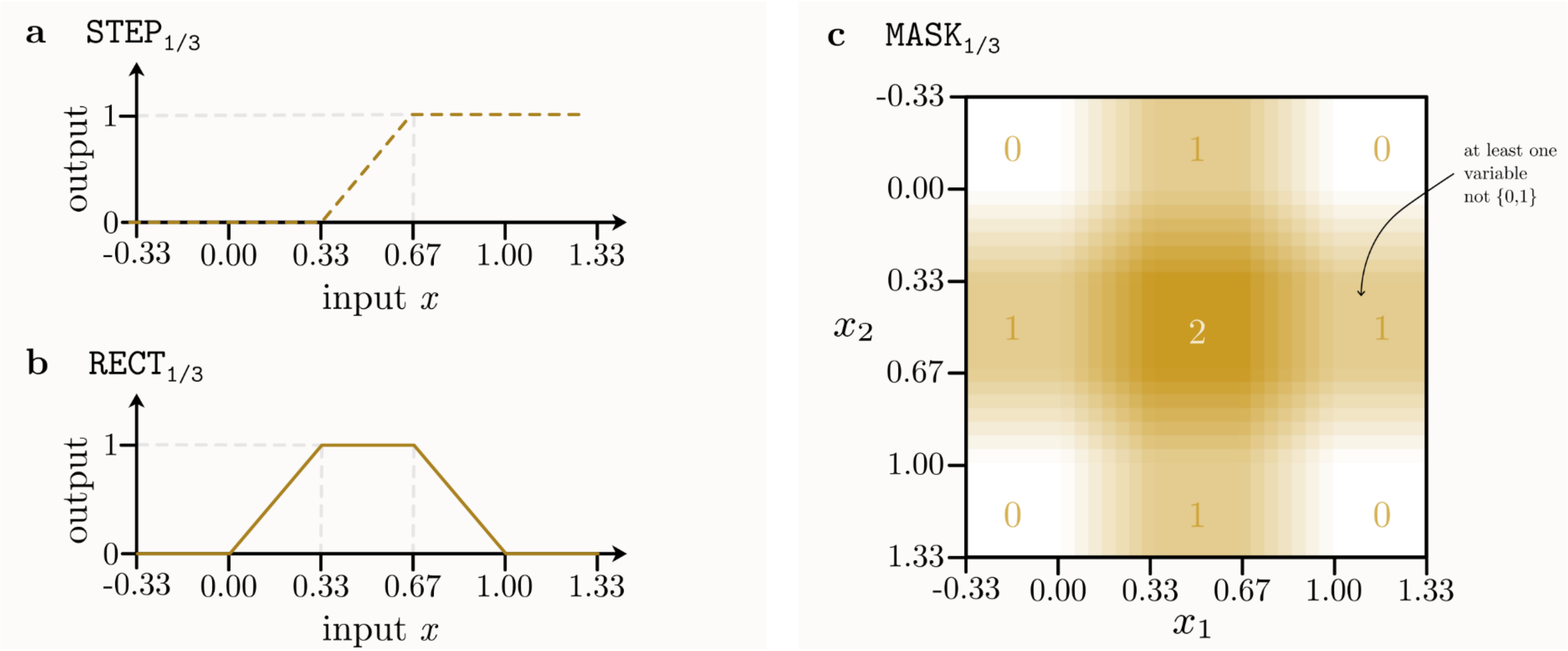
- Problem: we have to continuously connect these zero values on the multiwall with the correct non-zero values required at the unique binary point in each orthant, using only ReLU's and linear functions

- This smooth interpolation should not leak any information on the key



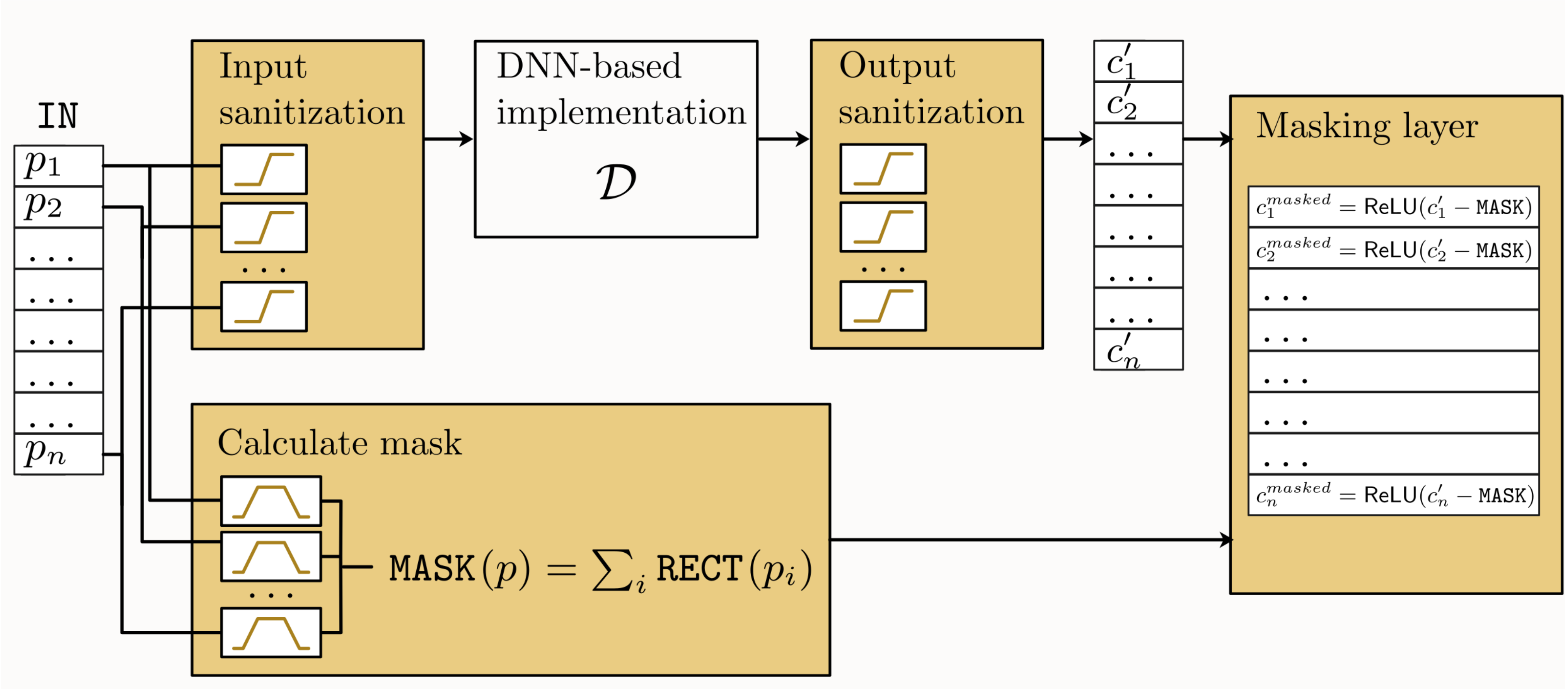
Third step: force all outputs for inputs in the “danger zone” to be identically zero

- In addition to the tighter STEP function, we introduce a new function RECT(x)=ReLU(x)-ReLU(x-0.33)-ReLU(x-0.66)+ReLU(x-1)
- We then define MASK(x1,…,xn) = ∑RECT(xi) for i=1,…,n



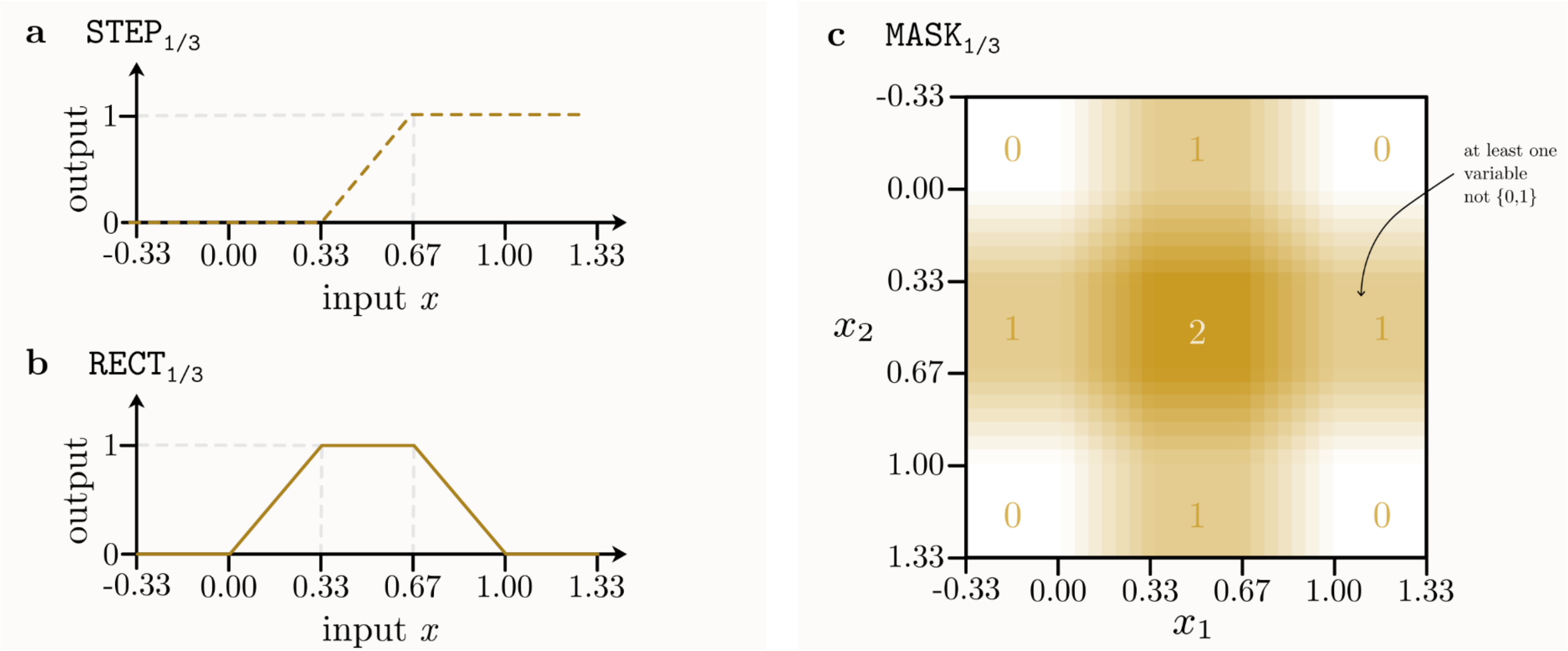
The final DNN implementation

- Combines all the previously defined filter functions, where each one of them is crucial



Third step: force all outputs for inputs in the “danger zone” to be identically zero

- MASK(x1,…,xn) is a smoothed continuous version of the multiwall
- It has a value of at least 1 at any point in the multiwall
- It has a value of 0 at any binary point in the input space (with just 0/1)



Why this DNN implementation is correct

- For any binary vector of 0/1 values, the initial input sanitization leaves the inputs unchanged
- The (potentially insecure) AES implementation then provides the correct 0/1 output values
- These outputs are again left unchanged by the final STEP sanitizations
- The sanitized outputs are not affected by the zero-valued MASK

Why this implementation is provably secure

- We want to show that real valued queries do not leak any information about the secret key that is not already leaked via binary valued queries to the primitive.

- Intuition: Any input within the danger zone yields only zero outputs

- For any input in a particular orthant which is not in the danger zone, the output is completely determined by the output of AES at the unique binary input contained in that orthant, interpolated smoothly by the MASK of the known values of the plaintext “bits”. This can be computed without any knowledge of the secret key bits!

The extra cost of securing a DNN implementation

- To obtain our secure DNN implementation of a cryptographic functionality, we can start with any (potentially insecure) DNN implementation such as the easily breakable natural implementation described above

- We can then make it secure by adding a constant number of additional layers and a linear number of additional ReLU-based neurons (as a function of the number of input and output values)

- This is a negligible cost for any nontrivial DNN, and thus our construction is very easy and completely practical

How to secure other cryptographic functionalities

- Consider, for example, the case of public key signature verification

- This functionality has no secret key, so our security guarantee (of not leaking any information about it) is meaningless

- The functionality should accept a message M and a signature S, and compute a function VERIFY(M,S) which should output 1 when the signature is valid and 0 when the signature is not valid.

- Given a DNN implementation of VERIFY, the attacker wins if he can produce some real-valued S’ which makes Verify(M,S’)=1

How to secure other cryptographic functionalities

- The security of the signature scheme in the binary case does not imply that its DNN implementation is also secure for real valued signatures

- We can use our sanitization techniques to force the output of VERIFY to be 0 for any real valued signature in the danger zone

- This makes our DNN implementation provably secure in the sense that any attacker which can find a real valued S’ satisfying the DNN version of VERIFY can also find a binary S satisfying the original (binary) version of VERIFY

Using other activation functions in the DNN

- So far we have assumed that the DNN uses the ReLU activation function. Can we apply our attacks to DNN's with other activation functions such as sigmoid(x) = 1/{1 + exp(-x)} ?
- The answer is yes, with some modifications
- Implementing XOR(x,k) with the sigmoid function:
  - Consider the function c1*{sigmoid(x-k+1) + sigmoid(k-x+1)} – c2
  - With a proper choice of the constants c1 and c2, we can make sure that for inputs (0,0) and (1,1) the outputs will be 0, and for (0,1) and (1,0) the output will be 1.
  - Once again, if we jitter x around 0, then when k=0 the output will change symmetrically, while when k=1 the output will change asymmetrically, so we can extract the key bit k by observing the values of the final output

Conclusions

- In this talk I defined the new research area of how to implement digital cryptography in an analog computer
- I defined the notion of natural implementation of schemes
- I demonstrated the insecurity of such natural implementations
- I described a different implementation which is provably secure