Building Blocks for LSTM Homomorphic Evaluation with TFHE

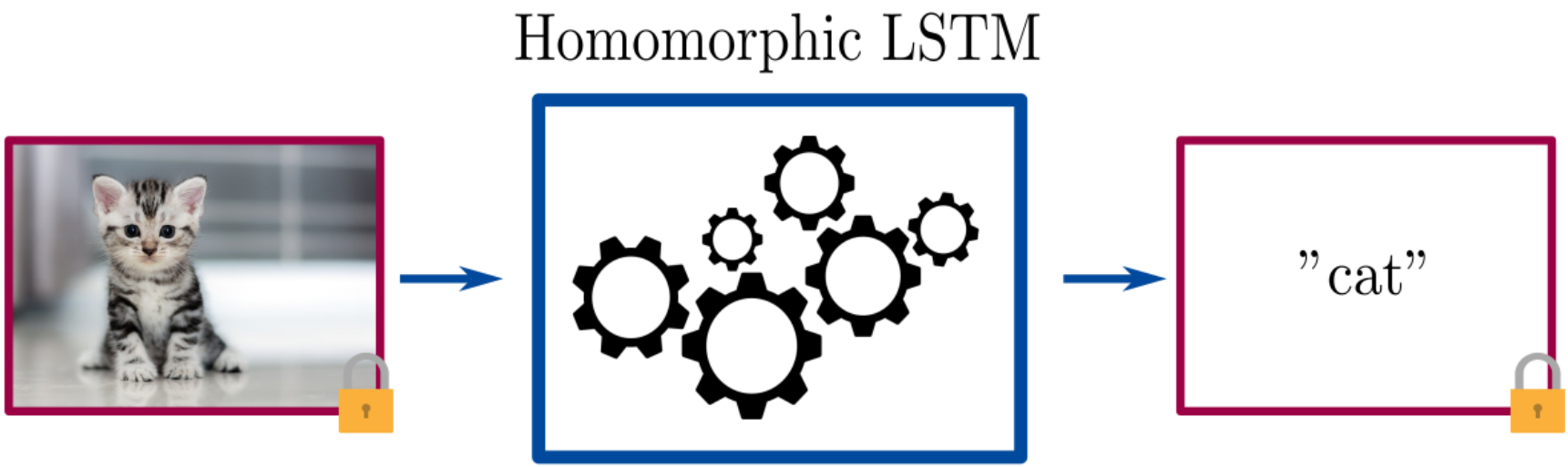
Daphné Trama Pierre-Emmanuel Clet Aymen Boudguiga
Renaud Sirdey

Université Paris-Saclay, CEA-List, Palaiseau, France

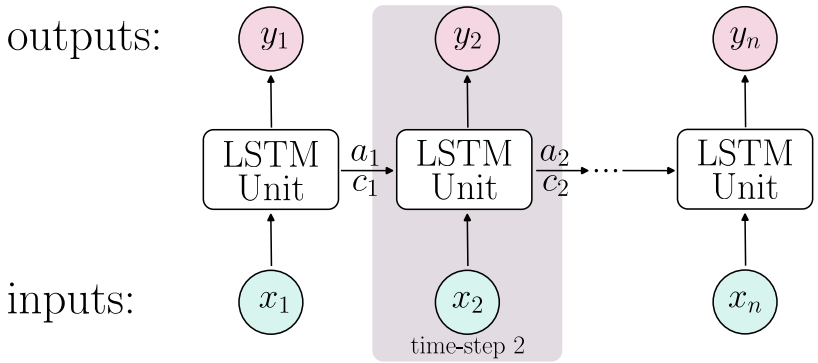
17 octobre 2023

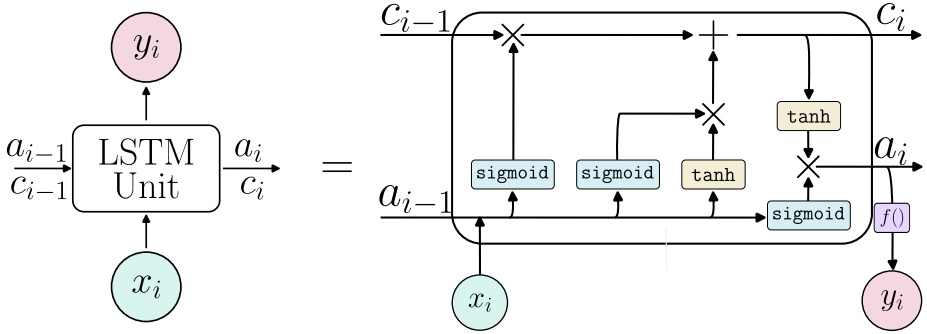
Long Short-Term Memory (LSTM)

- text translation
- feeling analysis
- speech analysis
- diagnosis findings on medical data
- genomic data analysis



Equations of the i^{th} LSTM cell





$$\tilde{c}^{\langle i \rangle} = \tanh(W_c \cdot [a^{\langle i-1 \rangle}, x^{\langle i \rangle}] + b_c)$$

$$\Gamma_u^{\langle i \rangle} = \sigma(W_u \cdot [a^{\langle i-1 \rangle}, x^{\langle i \rangle}] + b_u)$$

$$\Gamma_f^{\langle i \rangle} = \sigma(W_f \cdot [a^{\langle i-1 \rangle}, x^{\langle i \rangle}] + b_f)$$

$$\Gamma_o^{\langle i \rangle} = \sigma(W_o \cdot [a^{\langle i-1 \rangle}, x^{\langle i \rangle}] + b_o)$$

$$c^{\langle i \rangle} = \Gamma_u \cdot \tilde{c}^{\langle i \rangle} + \Gamma_f \cdot c^{\langle i-1 \rangle}$$

$$a^{\langle i \rangle} = \Gamma_o \cdot \tanh(c\langle i \rangle)$$

Choice of scheme

- non linear functions ☹
- polynomial approximations ! ☺
- BUT :
  - levelled schemes (BGV, BFV, CKKS, ...) : large multiplicative depth ⇒ large parameters ⇒ NOT EFFICIENT
    Or, if it exists, the bootstrapping takes at least several seconds
  - practicable bootstrapping-able schemes (TFHE) : decomposing every large input into digits in a basis B ⇒ at least one bootstrapping per simple operation ($+$ or $\times$) ⇒ TIME-CONSUMING
- a recent and rather unexplored solution : Look Up Tables (LUTs) by means of programmable bootstrapping

Scheme	Library	Number of Slots	Bits of Precision	Latency (sec)	Throughput slots/sec	Notes
CGGI TFHE	OpenFHE	1	1	0.058	17.2	Latency of NAND
			3-4	0.694	1.44	Latency of f(x) mod p
CKKS	OpenFHE	32,768	13	48.4	677	Full packing
			30	96.8	339	2 bootstrapping iterations to increase precision [BCC+22]
CKKS	OpenFHE	32	18	24.1	1.33	Sparse packing
			34	48.4	0.661	2 bootstrapping iterations to increase precision [BCC+22]
BGV	HElib	1,024	1	15	68.3	Thin bootstrapping [HS21]
			16	163	6.28	Full bootstrapping [HS21]

Benchmark from Badawi, A.A., Polyakov, Y.: Demystifying bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/149 (2023)

TFHE

TFHE security relies on **LWE** problem.
$\mathscr{M} = \{0, 1, \cdots, p-1\}$ a finite space of cleartexts of size p.
$\mathbb{T} = \mathbb{R}/\mathbb{Z}$

Encryption

$Enc(0) = c = (a, b) \in \mathrm{TLWE}_s(0)$ where :

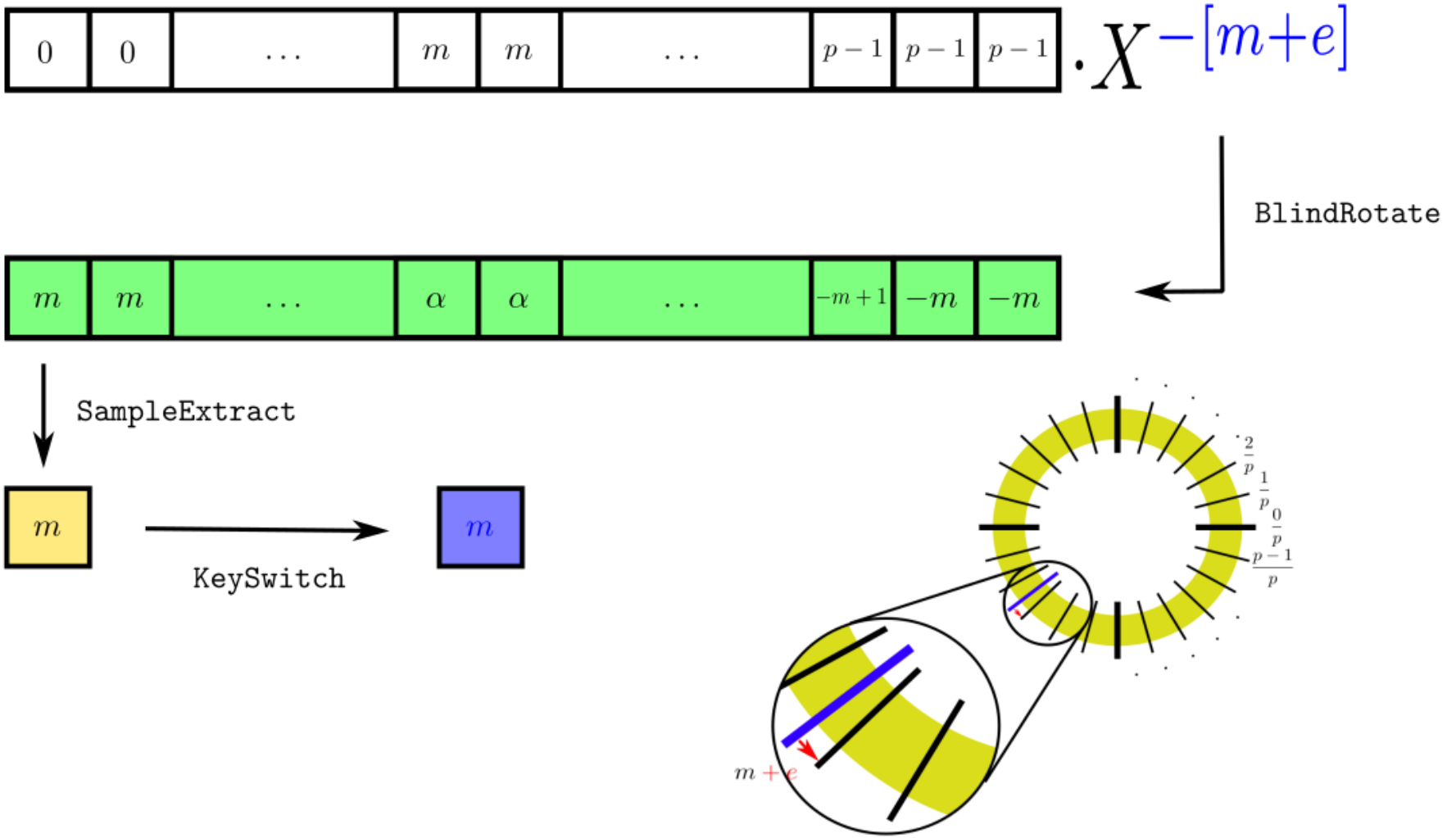
- a is uniformly sampled from $\mathbb{T}^n$
- $b = \langle a, s \rangle + e$
- the secret key s uniformly sampled from $\{0, 1\}^n$
- the error $e \in \mathbb{T}$ is sampled from a Gaussian distribution with mean 0

Then with $m \in \mathscr{M}$, $Enc(m) = [m] = c' = c + (0, m) = (a, b') = (a, b + m)$.

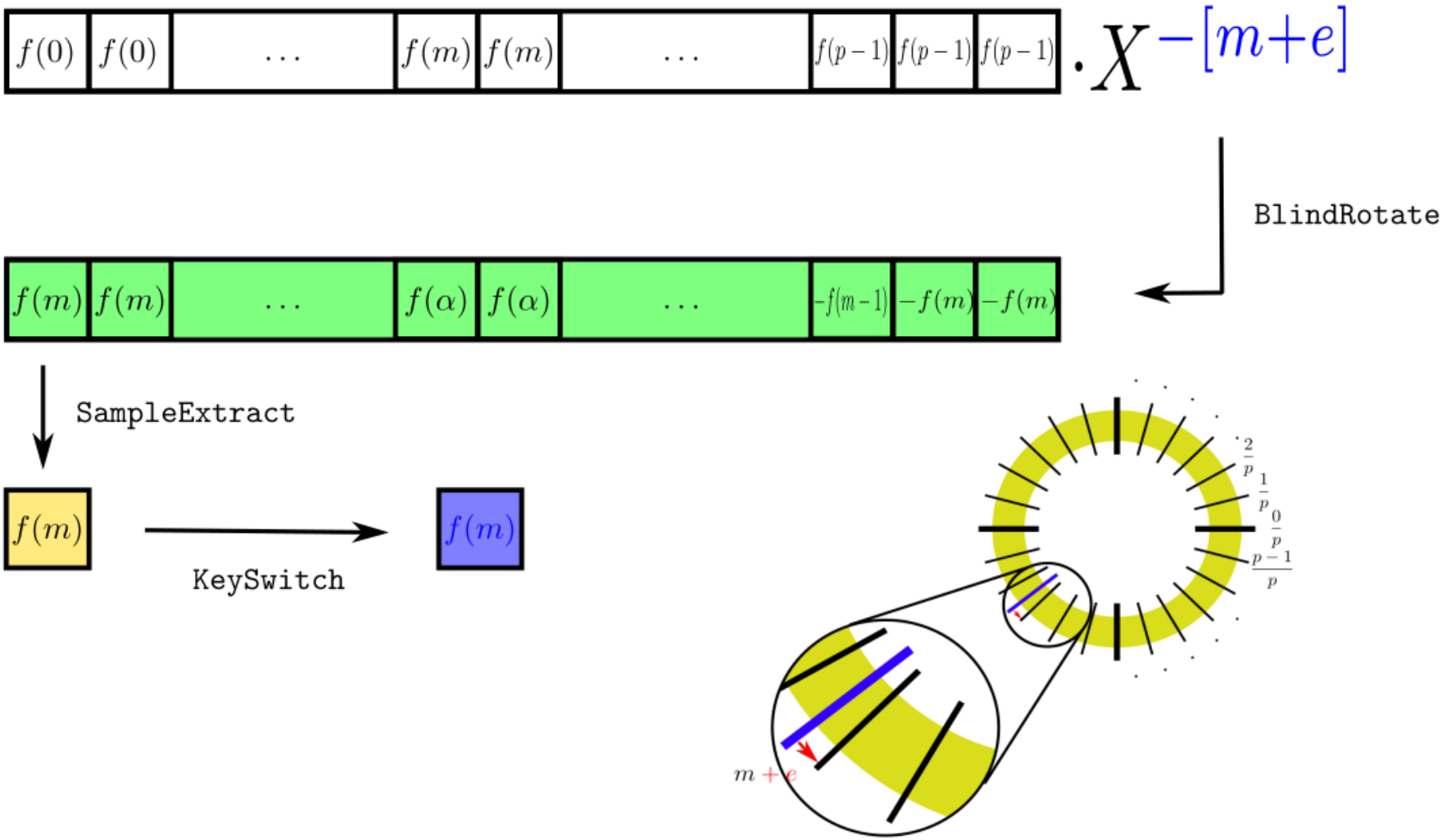
Decryption

$\phi(c') = b' - \langle a, s \rangle$
$Dec(c') = \lfloor \phi(c') \rceil = \lfloor b' - \langle a, s \rangle \rceil = \lfloor m + e \rceil = m$

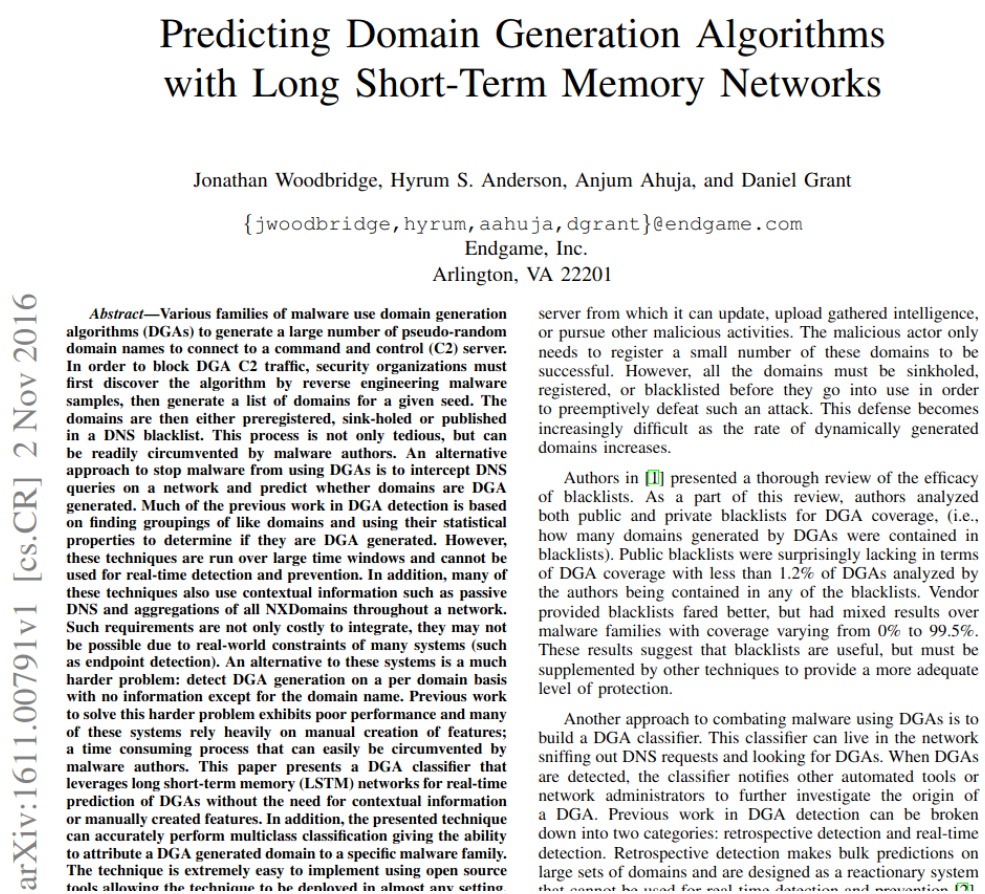
Bootstrapping



Functional Bootstrapping



Use Case



Predicting Domain Generation Algorithms with Long Short-Term Memory Networks

Jonathan Woodbridge, Hyrum S. Anderson, Anjum Ahuja, and Daniel Grant

{jwoodbridge,hyrum,aahuja,dgrant}@endgame.com
Endgame, Inc.
Arlington, VA 22201

***Abstract*—Various families of malware use domain generation algorithms (DGAs) to generate a large number of pseudo-random domain names to connect to a command and control (C2) server. In order to block DGA C2 traffic, security organizations must first discover the algorithm by reverse engineering malware samples, then generate a list of domains for a given seed. The domains are then either preregistered, sink-holed or published in a DNS blacklist. This process is not only tedious, but can be readily circumvented by malware authors. An alternative approach to stop malware from using DGAs is to intercept DNS queries on a network and predict whether domains are DGA generated. Much of the previous work in DGA detection is based on finding groupings of like domains and using their statistical properties to determine if they are DGA generated. However, these techniques are run over large time windows and cannot be used for real-time detection and prevention. In addition, many of these techniques also use contextual information such as passive DNS and aggregations of all NXDomains throughout a network. Such requirements are not only costly to integrate, they may not be possible due to real-world constraints of many systems (such as endpoint detection). An alternative to these systems is a much harder problem: detect DGA generation on a per domain basis with no information except for the domain name. Previous work to solve this harder problem exhibits poor performance and many of these systems rely heavily on manual creation of features; a time consuming process that can easily be circumvented by malware authors. This paper presents a DGA classifier that leverages long short-term memory (LSTM) networks for real-time prediction of DGAs without the need for contextual information or manually created features. In addition, the presented technique can accurately perform multiclass classification giving the ability to attribute a DGA generated domain to a specific malware family. The technique is extremely easy to implement using open source tools allowing the technique to be deployed in almost any setting.**

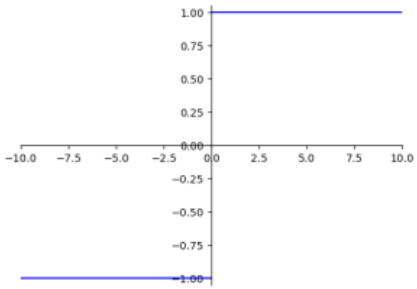
server from which it can update, upload gathered intelligence, or pursue other malicious activities. The malicious actor only needs to register a small number of these domains to be successful. However, all the domains must be sinkholed, registered, or blacklisted before they go into use in order to preemptively defeat such an attack. This defense becomes increasingly difficult as the rate of dynamically generated domains increases.

Authors in [1] presented a thorough review of the efficacy of blacklists. As a part of this review, authors analyzed both public and private blacklists for DGA coverage, (i.e., how many domains generated by DGAs were contained in blacklists). Public blacklists were surprisingly lacking in terms of DGA coverage with less than 1.2% of DGAs analyzed by the authors being contained in any of the blacklists. Vendor provided blacklists fared better, but had mixed results over malware families with coverage varying from 0% to 99.5%. These results suggest that blacklists are useful, but must be supplemented by other techniques to provide a more adequate level of protection.

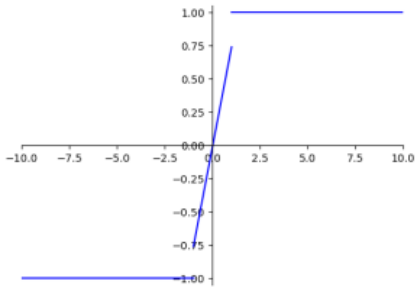
Another approach to combating malware using DGAs is to build a DGA classifier. This classifier can live in the network sniffing out DNS requests and looking for DGAs. When DGAs are detected, the classifier notifies other automated tools or network administrators to further investigate the origin of a DGA. Previous work in DGA detection can be broken down into two categories: retrospective detection and real-time detection. Retrospective detection makes bulk predictions on large sets of domains and are designed as a reactionary system that cannot be used for real-time detection and prevention. [2]

- « Predicting Domain Generation Algorithms using LSTMs », Jonathan Woodbridge, Hyrum S. Anderson, Anjum Ahuja et Daniel Grant
- LSTM with vector inputs of size 128.
- Network accuracy of 95.6%
- finite space $\mathscr{M}$ of size 16 (K-means clustering)

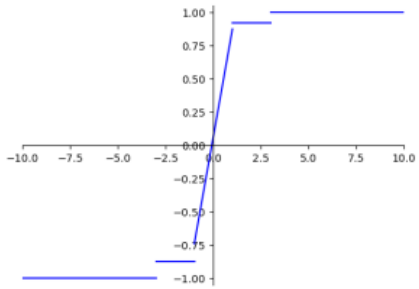
Discretisation



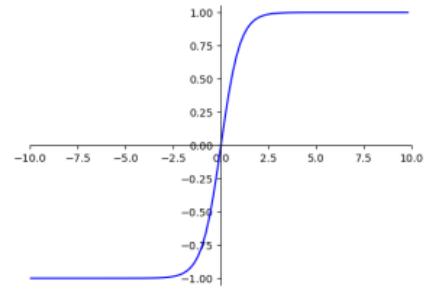
Functions `Sign` and `Heaviside`



Functions `3StepsSigmoid` and `3StepsTanh`



Our functions `StepSigmoid` ans `StepTanh`



Functions `Sigmoid` and `Tanh`

Used Functions	Accuracy of the LSTM
`Tanh` + `Sigmoid`	0.956
`Sign` + `Heaviside`	0.50
`3StepsTanh` + `3StepsSigmoid`	0.832
`StepTanh` + `StepSigmoid`	0.934

Accuracy of the network depending on the activation functions

Discretisation

$$\begin{aligned}\mathsf{StepSigmoid}(x) = & 0.13 \cdot \mathbb{1}_{]-6,-1]}(x) + \mathbb{1}_{]-1,1]}(x) \cdot (0.24x + 0.5) + 0.87 \cdot \mathbb{1}_{]1,6]}(x) \\ & + \cdot \mathbb{1}_{]6,\infty[}(x) \\ \mathsf{StepTanh}(x) = & -1 \cdot \mathbb{1}_{]-\infty,-3]}(x) - 0.875 \cdot \mathbb{1}_{]-3,-1]}(x) + \mathbb{1}_{]-1,1]}(x) \cdot (0.76x) \\ & + 0.92 \cdot \mathbb{1}_{]1,6]}(x) + \cdot \mathbb{1}_{]6,\infty[}(x).\end{aligned}$$

x	`StepSigmoid`(x)
]-∞, -6]	0
]-6, -1]	0.13
]-1, -0.834]	0.26
]-0.834, -0.668]	0.30
]-0.668, -0.501]	0.34
]-0.501, -0.335]	0.39
]-0.335, -0.169]	0.43
]-0.169, 0]	0.47
]0, 0.166]	0.52
]0.166, 0.332]	0.56
]0.332, 0.499]	0.60
]0.499, 0.665]	0.65
]0.665, 0.831]	0.69
]0.831, 1]	0.74
]1, 6]	0.87
]6, +∞]	1

x	`StepTanh`(x)
]-∞, -3]	-1
]-3, -1]	-0.875
]-1, -0.834]	-0.76
]-0.834, -0.668]	-0.622
]-0.668, -0.501]	-0.484
]-0.501, -0.335]	-0.346
]-0.335, -0.169]	-0.207
]-0.169, 0]	-0.069
]0, 0.166]	0.069
]0.166, 0.332]	0.207
]0.332, 0.499]	0.346
]0.499, 0.665]	0.484
]0.665, 0.831]	0.622
]0.831, 0.1]	0.76
]1, 3]	0.92
]3, +∞]	1

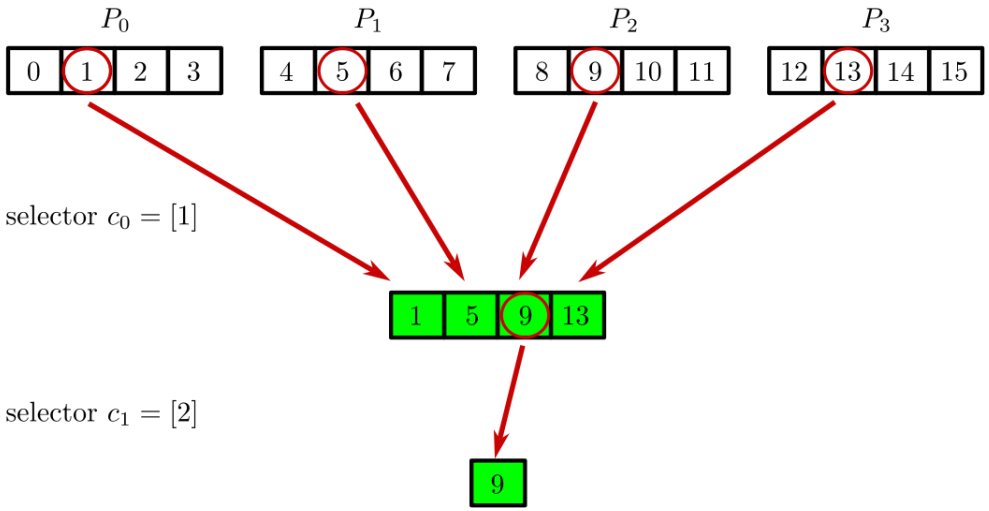
Our 16-steps `StepSigmoid` and `StepTanh`.

The Tree-based Method

Decomposition

$m = \sum_{i=0}^{d} m_i \cdot B^i$

$c = [m] = ([m_0], [m_1], \cdots, [m_{d-1}])$



Toy Example

- $B = 4$ and $d = 2$
- $m = 9 = 1 \cdot 4^0 + 2 \cdot 4^1$
- $c = ([1], [2]) = (c_0, c_1)$

$$P_0(X) = 0\cdot(1+X+\cdots X^{\frac{N}{4}-1})+1\cdot(X^{\frac{N}{4}}+\cdots X^{\frac{2N}{4}-1})+2\cdot(X^{\frac{2N}{4}}+\cdots X^{\frac{3N}{4}-1})+3\cdot(X^{\frac{3N}{4}}+\cdots X^{\frac{4N}{4}-1})$$

$$P_1(X) = 4\cdot(1+X+\cdots X^{\frac{N}{4}-1})+5\cdot(X^{\frac{N}{4}}+\cdots X^{\frac{2N}{4}-1})+6\cdot(X^{\frac{2N}{4}}+\cdots X^{\frac{3N}{4}-1})+7\cdot(X^{\frac{3N}{4}}+\cdots X^{\frac{4N}{4}-1})$$

$$P_2(X) = 8\cdot(1+X+\cdots X^{\frac{N}{4}-1})+9\cdot(X^{\frac{N}{4}}+\cdots X^{\frac{2N}{4}-1})+10\cdot(X^{\frac{2N}{4}}+\cdots X^{\frac{3N}{4}-1})+11\cdot(X^{\frac{3N}{4}}+\cdots X^{\frac{4N}{4}-1})$$

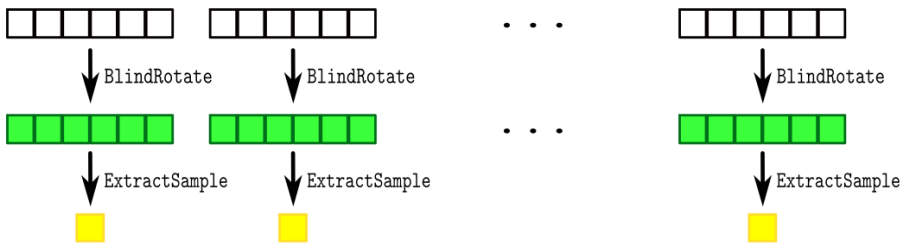
$$P_4(X) = 12\cdot(1+X+\cdots X^{\frac{N}{4}-1})+13\cdot(X^{\frac{N}{4}}+\cdots X^{\frac{2N}{4}-1})+14\cdot(X^{\frac{2N}{4}}+\cdots X^{\frac{3N}{4}-1})+15\cdot(X^{\frac{3N}{4}}+\cdots X^{\frac{4N}{4}-1})$$

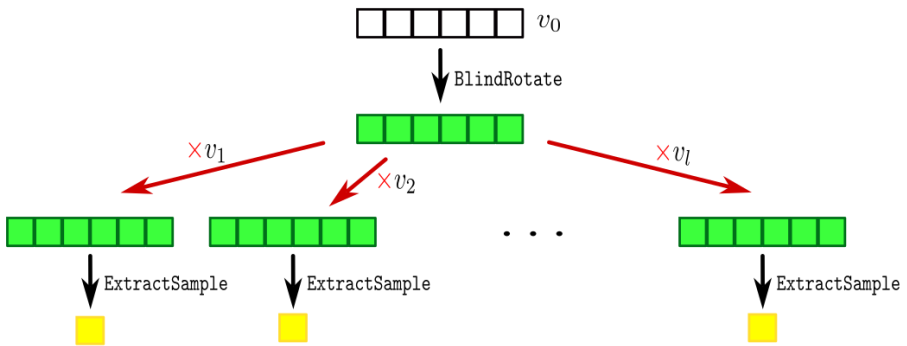
Multi-Value Bootstrapping (MVB)

Factorization

$$(1 + X + \cdots + X^{N-1}) \cdot (1 - X) \equiv 2 \bmod (X^N + 1)$$

$$\begin{aligned} P_{f_i} &= \frac{1}{2} \cdot (1 + X + \cdots + X^{N-1}) \cdot (1 - X) \cdot P_{f_i} \bmod (X^N + 1) \\ &= v_0 \cdot v_i \bmod (X^N + 1) \end{aligned}$$





Results

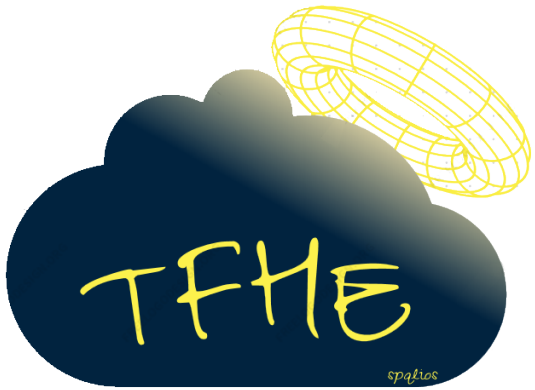
	number of bootstrapping		execution time	
	tree-based method	MVB optimization	tree-based method	MVB optimization
single `StepSigmoid`execution	5	2	0.28s	0.15s
complete LSTM unit execution	$5 \times 5 = 25$	$2 \times 5 = 10$	1.4s	0.75s
complete network execution	$128 \times 25 = 3200$	$128 \times 10 = 1280$	179s = 2min59s	96s = 1min36s

Execution time results (number of bootstrappings are provided for informational purposes, as bootstrapping is the most costly operation in TFHE).

$\rightsquigarrow$ network accuracy of 93.4%

Technical Setup

We run our code on a 4-core Intel Core i7-7600U 2.90GHz CPU (*with only one core activated*) and 16GiB total system memory with an Ubuntu 20.04.5 LTS server.



Key Take-aways

- LSTMs do not support rough approximations of their activation functions
  - do not use the pair (`Sign`, `Heaviside`)
- Discretization of the inputs and the internal coefficients of LSTMs (bias vectors and weight matrices) does not raise any issue with respect to network precision
- Our discretized activation functions let the accuracy of the network unchanged
- Our discretized activation functions are relatively efficient when evaluated over FHE (only a few hundredths of a second for one evaluation)
- The results are promising and open the door to an end-to-end TFHE evaluation of LSTMs with practical latencies !

Spoiler Alert

This method can be used in other applications, for instance, transciphering ! Here are some results of work on the "homomorphization" of the AES cipher (to be soon seen at WAHC) :

	execution time	time ratio
`i7-laptop` (1 thread)	4.5 mins = 270 secs	9. 4
`i7-laptop` (6 threads)	54.31 secs	1.9
`AMD-server` (1 thread)	5.7 mins = 342 secs	11.9
`AMD-server` (16 threads)	36.39 secs	1.3
`"i7-server"` (16 threads)	28.73 secs	1
Gentry et al.(1 thread)	18 mins	37.6
Mella and Susella (1 thread)	22 mins	45.9
Stracovsky et al. (16 threads)	4.2 mins = 252 secs	8.8

Thank you for your kind attention !

K-means Clustering for Homomorphic LSTM