

# Recent uses of Boolean and vectorial functions and related problems

**Claude Carlet** Universities of Bergen and Paris 8

# Outline

We present a chapter from a forthcoming book on Boolean and vectorial functions. This chapter is devoted to :

- ▶ Physical attacks and related problems on functions and codes
  - A new role of correlation immunity and of the dual distance of codes related to side channel attack (SCA) countermeasures,
  - Minimizing the number of nonlinear multiplications for reducing the cost of countermeasures against SCA,
  - Vectorial functions and threshold implementation,
  - Linear complementary dual codes and complementary pairs of codes used for direct sum masking,

- Robust codes, algebraic manipulation detection (AMD) codes.
- ▶ Fully homomorphic encryption (FHE), hybrid symmetric-FHE protocols for the cloud, and related questions on Boolean functions (with restricted inputs).
- ▶ Local pseudorandom generators (the Goldreich pseudorandom generator) and related criteria on Boolean functions.
- ▶ The Gowers norm on pseudo-Boolean functions.

## **Forthcoming book on Boolean and vectorial functions**

The new book is a reorganized and completed version of two chapters by C.C. in a CUP monography (Y. Crama and P. Hammer Eds.) :

Boolean Functions for Cryptography and Error Correcting Codes

Vectorial Boolean Functions for Cryptography

The new book is entitled :

**Boolean Functions for Cryptography and Coding Theory**

Since these chapters were written (in 2009), about 1500 papers have been published in this domain.

New notions on Boolean and vectorial functions and new ways of using them have also emerged.

In this talk, we present the chapter devoted to these recent and/or not enough studied directions of research.

**Tentative table of content of the new book :**

<b>1</b>	<b>Introduction to cryptography, codes, Boolean and vectorial functions</b>	<i>page</i> 15
1.1	Cryptography	15
1.1.1	Symmetric versus public-key cryptosystems	16
1.1.2	Block ciphers versus stream ciphers	17
1.2	Error correcting codes	18
1.2.1	Detecting and correcting capacities of a code	19
1.2.2	Parameters of a code	21
1.2.3	Linear codes	21
1.2.4	Cyclic codes	25
1.2.5	The MacWilliams identity and the notion of dual distance	29
1.3	Boolean functions	33
1.3.1	Boolean functions and stream ciphers	34
1.3.2	Boolean functions and error correcting codes	39
1.4	Vectorial functions	40
1.4.1	Vectorial functions and stream ciphers	40
1.4.2	Vectorial functions and block ciphers	40
1.4.3	Vectorial functions and error correcting codes	42
<b>2</b>	<b>Generalities on Boolean and vectorial functions</b>	43
2.1	A hierarchy of equivalence relations over Boolean and vectorial functions	43
2.1.1	Relations between these equivalences	45
2.2	Representations of Boolean functions and vectorial functions	46
2.2.1	Algebraic normal form	47
2.2.2	Univariate and trace representations	57
2.2.3	Bivariate representation of functions with even number of input bits	63
2.2.4	Representation over the reals (numerical normal form)	64
2.3	The Fourier-Hadamard transform and the Walsh transform	69
2.3.1	Fourier-Hadamard transform of pseudo-Boolean functions	69
2.3.2	Fourier-Hadamard and Walsh transforms of Boolean functions	71

2.3.3	Properties of the Fourier-Hadamard and Walsh transforms of Boolean functions	74
2.3.4	Fourier-Hadamard transform and numerical normal form	83
2.3.5	The size of the support of the Fourier-Hadamard transform and Cayley graphs	87
2.3.6	The Walsh transform of vectorial functions	88
2.3.7	The multidimensional Walsh transform	91
2.4	Fast computation of S-boxes	92
<b>3</b>	<b>Boolean functions, vectorial functions and cryptography</b>	<b>93</b>
3.1	Cryptographic criteria (and related parameters) for Boolean functions	93
3.1.1	Balancedness	94
3.1.2	Algebraic degree	94
3.1.3	Nonlinearity and higher order nonlinearity	95
3.1.4	Correlation immunity and resiliency	103
3.1.5	Algebraic immunity and fast algebraic immunity	106
3.1.6	Variants to these criteria in relationship with guess and determine attacks	114
3.1.7	Avalanche criteria, inexistence of nonzero linear structure, correlation with subsets of indices	114
3.1.8	Complexity parameters	120
3.2	Cryptographic criteria for vectorial functions in stream and block ciphers	130
3.2.1	Balancedness of vectorial functions	130
3.2.2	Algebraic degree of vectorial functions	133
3.2.3	Nonlinearity of vectorial functions	134
3.2.4	Algebraic immunity of vectorial functions	146
3.3	Cryptographic criteria and parameters for vectorial functions in stream ciphers	149
3.3.1	Correlation immunity and resiliency of vectorial functions	150
3.3.2	Unrestricted nonlinearity of vectorial functions	152
3.4	Cryptographic criteria and parameters for vectorial functions in block ciphers	156
3.4.1	Differential uniformity	156
3.4.2	Other features also related to attacks	165
3.5	Search for functions achieving the desired features	165
3.5.1	The difficulty of designing good S-boxes	165
3.5.2	Constructions versus computer investigations of Boolean and vectorial functions	166
3.6	Boolean and vectorial functions for secret sharing, authentication, diffusion	168
3.6.1	Secret sharing, access structures and minimal codes	168
3.6.2	Authentication schemes	173

<b>4</b>	<b>Boolean functions, vectorial functions and error correcting codes</b>	174
4.1	Reed-Muller codes	174
4.1.1	Minimum distance and minimum weight codewords	176
4.1.2	Dual	177
4.1.3	Automorphism group	178
4.1.4	Cyclicity of the punctured code $R^*(r, n)$	178
4.1.5	The problem of determining the weight distributions of Reed-Muller codes	179
4.1.6	Covering radius	180
4.2	Other codes related to Boolean functions	182
4.2.1	Linear codes	182
4.2.2	Unrestricted codes	183
4.2.3	Codes and diffusion layers in block ciphers	184
4.2.4	Codes and association schemes	185
4.2.5	Codes and secret sharing	185
<b>5</b>	<b>Functions with weights, Walsh spectra and nonlinearities easier to study</b>	186
5.1	Affine functions and their combinations	186
5.1.1	Affine functions	186
5.1.2	Maierana-McFarland functions	186
5.1.3	Niho and $\mathcal{PS}_{ap}$ functions	189
5.2	Quadratic functions and their combinations	192
5.2.1	Quadratic Boolean functions	192
5.2.2	Concatenations of quadratic functions	201
5.3	Cubic functions	202
5.4	Indicators of flats	203
5.4.1	Concatenations of sums of indicators of flats and affine functions	204
5.5	Normal functions	204
5.6	Functions admitting (partial) covering sequences	205
5.6.1	The case of Boolean functions	205
5.6.2	The case of vectorial functions	208
5.7	Functions with low univariate degree	211
<b>6</b>	<b>Bent functions and plateaued functions</b>	212
6.1	Bent Boolean functions	213
6.1.1	Extended affine invariance of bentness and automorphism group of a function	215
6.1.2	Characterization of bentness by the derivatives	215
6.1.3	Characterization of bentness by power moments of the Walsh transform	216
6.1.4	Characterization of bentness by the $NNF$	218



6.1.5	Characterization of bentness by codes	219
6.1.6	Characterization of bentness by difference sets, relative difference sets and structures of finite geometries	220
6.1.7	Bent Boolean functions and designs	221
6.1.8	The dual of a bent Boolean function	221
6.1.9	Bound on algebraic degree and related properties	225
6.1.10	Bent Boolean functions and affine subspaces	226
6.1.11	Affine spaces of bent Boolean functions	227
6.1.12	The graph of bent functions	229
6.1.13	Bent Boolean functions of low algebraic degrees	229
6.1.14	Bent Boolean functions in few variables	232
6.1.15	Primary constructions of bent Boolean functions	233
6.1.16	Secondary constructions of bent Boolean functions	259
6.1.17	Decompositions of bent functions	267
6.1.18	Class $\mathcal{GPS}$ and a geometric characterization of bent Boolean functions	268
6.1.19	On the number of bent Boolean functions	269
6.1.20	Hyper-bent, homogeneous, symmetric and rotation symmetric bent Boolean functions	271
6.1.21	Normal and non-normal bent Boolean functions	279
6.1.22	Kerdock codes	281
6.2	Partially-bent and plateaued Boolean functions	283
6.2.1	Partially-bent functions	283
6.2.2	Plateaued Boolean functions	286
6.2.3	Characterizations of plateaued Boolean functions	287
6.2.4	The subclasses of semi-bent and near-bent functions	290
6.2.5	Primary constructions of plateaued Boolean functions	291
6.2.6	Secondary constructions of plateaued Boolean functions	293
6.3	Bent <sub>4</sub> and partially-bent <sub>4</sub> functions	294
6.4	Bent vectorial functions	296
6.4.1	Primary constructions of bent vectorial functions	297
6.4.2	Secondary constructions of bent vectorial functions	300
6.5	Plateaued vectorial functions	302
6.5.1	Characterizations of plateaued vectorial functions	303
6.5.2	CCZ and EA equivalence of plateaued functions	309
6.5.3	Constructions of plateaued vectorial functions	309
<b>7</b>	<b>Correlation immune and resilient functions</b>	<b>311</b>
7.1	Correlation immune and resilient Boolean functions	311
7.1.1	Bound on the correlation immunity order	312
7.1.2	Bounds on algebraic degree	312
7.1.3	Bounds on the nonlinearity	314
7.1.4	Bound on the maximum correlation with index subsets	317
7.1.5	Relationship with other criteria	317

7.1.6	Relationship with covering sequences	318
7.1.7	Primary constructions of correlation immune and resilient functions	318
7.1.8	Secondary constructions of correlation immune and resilient functions	325
7.1.9	On the number of correlation immune and resilient functions	339
7.2	Resilient vectorial Boolean functions	341
7.2.1	Constructions of resilient vectorial Boolean functions	342
<b>8</b>	<b>Functions satisfying SAC, PC, EPC, or having good GAC</b>	<b>347</b>
8.1	$PC(l)$ criterion	347
8.1.1	Characterizations	348
8.1.2	Constructions	348
8.2	$PC(l)$ of order $k$ and $EPC(l)$ of order $k$ criteria	349
8.3	Functions having good sum-of-squares indicator and/or good absolute indicator	350
<b>9</b>	<b>Algebraic immune functions</b>	<b>351</b>
9.1	Algebraic immune Boolean functions	351
9.1.1	General properties of the algebraic immunity and its relationship with some other criteria	354
9.1.2	The problem of finding functions achieving high algebraic immunity and high nonlinearity	365
9.1.3	The functions with high algebraic immunity found so far and their parameters	366
9.1.4	Secondary constructions of algebraic immune functions	373
9.1.5	Another direction of research of Boolean functions suitable for stream ciphers	376
9.1.6	An additional condition modifying the study of Boolean functions for stream ciphers	377
9.2	Algebraic immune vectorial functions	377
9.2.1	Known bounds on algebraic immunities	379
9.2.2	Bounds on the numbers $d_{n,m}$ and $D_{n,m}$	379
9.2.3	Consequences on the number of output bits and on the tightness of the bounds	381
9.2.4	Nonlinearity and higher order nonlinearity	382
9.2.5	Constructions of algebraic immune vectorial functions	383
<b>10</b>	<b>Particular classes of Boolean functions</b>	<b>385</b>
10.1	Symmetric functions	385
10.1.1	Representation	385
10.1.2	Fourier-Hadamard and Walsh transforms	387
10.1.3	Nonlinearity	388
10.1.4	Resiliency	390

10.1.5	Algebraic immunity and fast algebraic immunity	391
10.1.6	The subclass of threshold functions	392
10.2	Rotation symmetric, idempotent and other similar functions	395
10.3	Direct sums of monomials	397
10.3.1	Triangular functions	398
10.4	Monotone functions	398
<b>11</b>	<b>Highly nonlinear vectorial functions with low differential uniformity</b>	<b>403</b>
11.1	The covering radius bound; bent/perfect nonlinear functions	404
11.2	The Sidelnikov-Chabaud-Vaudenay bound	405
11.3	Almost perfect nonlinear and almost bent functions	405
11.3.1	Characterizations of AB and APN functions	407
11.3.2	The particular case of <i>power functions</i>	416
11.3.3	Componentwise APNness (CAPNness)	423
11.3.4	Plateaued APN functions	424
11.4	The known infinite classes of AB functions	427
11.4.1	Power AB functions	427
11.4.2	Non-power AB functions	429
11.5	The known infinite classes of APN functions	432
11.5.1	Sporadic APN (and AB) functions	432
11.5.2	Power APN functions	432
11.5.3	Non-power APN functions	437
11.5.4	The extended Walsh spectra of known APN functions	445
11.5.5	Conclusion on known APN functions	445
11.6	Differentially uniform functions	446
11.6.1	Characterizations by the Walsh transform	446
11.6.2	Componentwise Walsh uniformity (CWU)	448
11.6.3	Cyclic difference sets, cyclic-additive difference sets and the CWU property	448
11.6.4	The known differentially 4-uniform $(n, n)$ -permutations, $n$ even	450
11.6.5	Other differentially 4-uniform $(n, n)$ -functions	455
11.6.6	Other Differentially uniform $(n, n)$ -functions	456
11.6.7	On the best differential uniformity of $(n, m)$ -functions	456
<b>12</b>	<b>Recent uses and problems on Boolean and vectorial functions</b>	<b>458</b>
12.1	Physical attacks and related problems on functions and codes	458
12.1.1	A new role of correlation immunity and of the dual distance of codes related to side channel attack countermeasures	465
12.1.2	Vectorial functions in univariate form: minimizing the number of nonlinear multiplications for reducing the cost of countermeasures	467
12.1.3	Vectorial functions and algebraic side channel attacks	470

12.1.4	Vectorial functions and threshold implementation	470
12.1.5	Linear complementary dual codes and complementary pairs of codes used for direct sum masking	479
12.1.6	Robust codes, algebraic manipulation detection (AMD) codes and vectorial functions	482
12.2	Fully homomorphic encryption and related questions on Boolean functions	489
12.2.1	The FLIP cipher	490
12.2.2	Boolean functions with restricted inputs	492
12.3	Local pseudorandom generators and related criteria on Boolean functions	504
12.3.1	The Goldreich pseudorandom generator	504
12.4	The Gowers norm on pseudo-Boolean functions	506
<b>13</b>	<b>Open questions</b>	512
13.1	Questions of general cryptography dealing with functions	512
13.2	General questions on Boolean functions and vectorial functions	512
13.3	Bent functions and plateaued functions	513
13.4	Correlation immune and resilient functions	514
13.5	Algebraic immune functions	514
13.6	Highly nonlinear vectorial functions with low differential uniformity	515
13.7	Recent uses and problems on Boolean and vectorial functions	516
	<i>References</i>	517
<b>14</b>	<b>Appendix: finite fields</b>	588
14.1	Prime fields and fields with 4, 8 and 9 elements	588
14.1.1	Characteristic of a finite field	588
14.1.2	Prime fields	589
14.1.3	Possible size of a finite field	589
14.1.4	Extending prime fields; fields with 4, 8 and 9 elements	590
14.2	General finite fields. Construction of finite fields, primitive element	591
14.2.1	The fundamental equation over finite fields	592
14.2.2	Existence of finite fields	593
14.2.3	Uniqueness of finite fields	594
14.2.4	Frobenius automorphism	595
14.2.5	Primitive element	595
14.3	Representation (additive and multiplicative) ; trace function	596
14.3.1	Trace function	597
14.3.2	Subfields and other trace functions	598
14.4	Permutations on a finite field	599
14.4.1	Examples of permutation polynomials	599
14.4.2	General results on permutation polynomials	601
14.5	Equations over finite fields	602
	<i>Index</i>	606

# Physical attacks and related problems on functions and codes

The implementation of cryptographic algorithms in embedded devices leaks information on the data manipulated by the algorithm, leading to *side channel attacks* (SCA).

The attacker model is then not a black box but a gray box.

This information can be *traces* of electromagnetic emanations, power consumption, photonic emission...

SCA are very powerful on block ciphers if countermeasures are not included in the implementation of the cryptosystems.

A *sensitive variable*  $Z$  is chosen in the algorithm, whose value is stored in a *register* and depends on the plaintext and a few key bits.

The register *leaks*.

The emanations from the register are measured. They disclose a noisy version of a real-valued function  $\mathcal{L}$  of the sensitive variable, for instance the Hamming weight of  $Z$ .

A statistical method finds then the value of the key bits which optimizes the correlation between the traces and a *modeled leakage*.

The original implementation of the AES can be attacked this way in a few seconds with a few traces.

*Counter-measures exist.*

Most common : *mask* each sensitive variable  $Z$  by splitting it.

- 2 shares :  $Z \oplus M \quad || \quad M$ , where  $M$  is drawn at random.

*For going through boxes :*

In hardware (FPGA, ASIC, ...) : use memory avoiding leak.

In software (smart cards) : transform every function  $x \mapsto F(x)$  in the algorithm into a function  $F' : (m_0, m_1) \mapsto (m'_0, m'_1)$  such that :

$$m'_0 + m'_1 = F(m_0 + m_1)$$

and the knowledge of one intermediate variable does not give any information on  $x$ .

Such  $F'$  is called a *masked version* of  $F$ .



Masking linear functions is costless but masking S-boxes has a cost.

In software applications (smart cards), masking the algorithm can multiply by more than 20 the execution time.

In hardware applications (ASIC, FPGA), the implementation area is roughly tripled.

- The counter-measure of masking with a single mask (i.e. two shares) cannot resist *Higher order SCA* (HO-SCA).

Higher order masking :

$d + 1$  shares :  $M_1, \dots, M_d$  are chosen at random and

$$M_{d+1} = Z \oplus M_1 \oplus \dots \oplus M_d.$$

The complexity of the HO-SCA attack (in time and in the number of traces) is exponential in the order :  $O(V^d)$ , where  $V$  is the variance of the noise.

The cost in running time and memory is quadratic in  $d$ .

Hence, theoretically, the designer can take advantage over the attacker.

But the implementation must be efficient today while the SCA can be performed in the future ( $\rightarrow$  advantage for the attacker).

Hence it is important to be able to implement high order masking and therefore to reduce the cost of counter-measures against SCA.

► **A new role of correlation immunity and of the dual distance of codes related to side channel attack (SCA) countermeasures**

*Rotating S-boxes Masking (RSM, hardware) :*

to avoid leakage, the mask  $M$  is not processed at all.

Instead, the computation for the next S-box is done with a Look-Up-Table (LUT) of the masked S-box  $S'(x) = S(x \oplus M) \oplus M'$ .

This allows a perfect protection against SCA.

But having a LUT for each masked version of each S-box is not possible for reasons of memory.

A small number of S-boxes are then embedded already masked in the implementation.

At every encryption, the allocation of the S-box is random.

The countermeasure resists the  $d$ -th order attack if and only if the indicator  $f$  of the mask set satisfies

$$\forall a \in \mathbb{F}_2^n, 1 \leq w_H(a) \leq d \Rightarrow \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) + a \cdot x} = 0.$$

Equivalently, the indicator of  $\mathcal{M}$  is a  $d$ -CI function, that is,  $\mathcal{M}$  is a code of dual distance at least  $d + 1$ .

For  $d$  as large as possible, we look for such functions of *minimum nonzero Hamming weight*, since the lower the weight of this function, the cheaper the countermeasure.

- ▶ **Vectorial functions in univariate form : minimizing the number of nonlinear multiplications for reducing the cost of countermeasures**

The complexity of *masking* additions and linear multiplications (like  $x \times x$ ) is negligible compared to that of masking nonlinear multiplications.

We need to minimize the *masking complexity* of each S-box : the number of nonlinear multiplications needed to implement it.

For power functions  $F(x) = x^d$ , minimizing the number of nonlinear multiplications results in minimizing addition chains in a group.

The *inverse function*  $x \rightarrow x^{254} = x^{-1}$  in  $\mathbb{F}_{2^8}$  can be implemented with 4 nonlinear multiplications.

Most recent methods for general functions :

— The *Coron-Roy-Vivek (CRV) method* :

- starts with a union  $\mathcal{C}$  of cyclotomic classes  $\mathcal{C}_i$  in  $\mathbb{Z}/(2^n - 1)\mathbb{Z}$ ,
- the set of corresponding monomials  $x^j$  spans a subspace  $\mathcal{P}$  of  $\mathbb{F}_{2^n}[x]$ .
- $r$  polynomials  $P_1(x), \dots, P_r(x)$  are chosen in  $\mathcal{P}$  and  $r + 1$  polynomials  $P_{r+1}(x), \dots, P_{2r+1}(x)$  are searched in  $\mathcal{P}$  such that :

$$P(x) = \sum_{i=1}^r P_i(x) \times P_{r+i}(x) + P_{2r+1}(x) .$$

This method works heuristically.

— The *CPRR method* :

- starts by deriving a family of generators :

$$\begin{cases} G_1(x) = F_1(x) \\ G_i(x) = F_i(G_{i-1}(x)) \end{cases}$$
 where the  $F_i$  are random polynomials of algebraic degree  $s$ .

- randomly generates  $t$  polynomials  $Q_i = \sum_{j=1}^r L_j \circ G_j$ , where the  $L_j$  are linearized polynomials.

- searches for  $t$  polynomials  $P_i$  of algebraic degree  $s$  and for  $r + 1$  linearized polynomials  $L_i$  such that :

$$P(x) = \sum_{i=1}^t P_i(Q_i(x)) + \sum_{i=1}^r L_i(G_i(x)) + L_0(x) .$$



For masking  $P(x)$ , we use that for any function  $F$  of algebraic degree at most  $s$  :

$$F\left(\sum_{i=1}^d a_i\right) = \sum_{j=0}^s \mu_{d,s}(j) \sum_{\substack{I \subseteq \{1, \dots, d\} \\ |I|=j}} F\left(\sum_{i \in I} a_i\right) ,$$

where  $\mu_{d,s}(j) = \binom{d-j-1}{s-j} \bmod 2$  for every  $j \leq s$ .

▶ Vectorial functions and threshold implementation

Masking is efficient only if the leakage has some regularity.

In particular, hardware *glitches*, common in CMOS technology, change the leaking into functions  $\mathcal{L}$  having numerical degree larger than 1, because of the interactions between bits that they cause.

Glitch-free hardware is very expensive.

- A way of addressing glitches is the so-called *polynomial masking*, based on *multiparty computation*.

The masking operation is based on Shamir's secret sharing.

Not quite practical.

- Another S-box masking method, also based on ideas of multiparty computation, is *threshold implementation (TI)* :  
 — each input variable  $x_i$  is masked into

$$\mathbf{x}_i = (x_{i,1}, \dots, x_{i,t+1}) \in \mathbb{F}_2^{t+1}.$$

We have  $s(\mathbf{x}_i) = x_{i,1} \oplus \dots \oplus x_{i,t+1} = x_i$ .

Extending  $s$  to a function over  $\mathbb{F}_2^{(t+1)n}$ , we have then

$$s(\mathbf{x}) = x, \forall x = (x_1, \dots, x_n).$$

A  $t$ -realization of  $F$  is

$$\mathbf{F} = (F_1, \dots, F_{t+1}) : \mathbb{F}_2^{(t+1)n} \mapsto \mathbb{F}_2^{(t+1)m} \text{ such that :}$$

- *Correctness* : if  $x = s(\mathbf{x})$ , then  $F(x) = s(\mathbf{F}(\mathbf{x}))$ .
- *Non-completeness* : every  $F_j$  is independent of the  $j$ -th coordinate of each  $\mathbf{x}_i$ .
- *Uniformity* : for every  $\mathbf{b} = (b_1, b_2, \dots, b_{t+1})$  in  $\mathbb{F}_2^{(t+1)m}$ , we have :

$$|\{\mathbf{x} \in \mathbb{F}_2^{(t+1)n}; \mathbf{F}(\mathbf{x}) = \mathbf{b}\}| = 2^{t(n-m)} \times |\{x \in \mathbb{F}_2^n; F(x) = s(\mathbf{b})\}|$$

(if  $F$  is a permutation then  $\mathbf{F}$  is a permutation).

This property is needed to compose several TI's.

It is the difficult one to achieve !

Indeed, if  $d_{alg}(F) \leq t$ , then replacing each  $x_i$  in  $F(x)$  by the sum  $x_{i,1} \oplus \cdots \oplus x_{i,t+1}$  and storing in  $F_j$  all those monomials with indices different from  $j$ , we ensure correctness and non-completeness.

*Conversely :*

**Proposition 1.** *Let  $F$  be any  $(n, m)$ -function admitting a  $t$ -mask (i.e. a  $(t + 1)$ -share) TI with or without uniformity. Then the algebraic degree of  $F$  is at most  $t$ .*

For instance, the inverse function  $F(x) = x^{2^n-2}$  cannot have an  $(n - 1)$ -share (with  $(n - 2)$ -masks) TI.

Even for *quadratic functions*, there does not always exist a TI with uniformity of minimum number of masks (that is, with 2 masks).

The TI cost of a function increases exponentially with its degree.

This drawback can be bypassed by expressing functions as the compositions of lower algebraic degree functions.

Uniformity is ensured by introducing fresh randomness.

But randomness is costly too. So more work on TI is needed.

- ▶ Linear complementary dual codes and complementary pairs of codes used for direct sum masking

Direct sum masking consists in :

- encoding the sensitive data, say  $x \in \mathbb{F}_2^k$ , into a codeword of a  $k$ -dimensional linear subcode  $C$  of  $\mathbb{F}_2^n$ ,
- encoding the mask  $y$  drawn at random in  $\mathbb{F}_2^{n-k}$  into a codeword of an  $(n - k)$ -dimensional linear subcode  $D$  of  $\mathbb{F}_2^n$ .

The masked version of  $x$  equals then the sum of these two codewords. If  $G$  is a generator matrix of  $C$  and  $G'$  a generator matrix of  $D$ , we take then :

$$z = x \times G + y \times G'.$$

For allowing the final demasking at the end of the computation,  $C$  and  $D$  must have trivial intersection, that is, be supplementary :

$$\mathbb{F}_2^n = C \oplus D.$$

Every vector  $z \in \mathbb{F}_2^n$  can then be written in a unique way as

$$z = x \times G + y \times G'; \quad x \in \mathbb{F}_2^k, y \in \mathbb{F}_2^{n-k}.$$

$d$ -th order masking and another known method called inner product masking are particular cases of DSM.

Contrary to these other methods, it can be also a countermeasure against FIA.



A pair  $(C, D)$  of supplementary codes is called a *linear complementary pair (LCP)* of codes.

If the leak  $\mathcal{L}$  as a *pseudo-Boolean* function has numerical degree 1, the encoding with an LCP of codes  $(C, D)$  protects against :

- $d$ -th order HO-SCA if and only if the dual distance of  $D$  satisfies  $d(D^\perp) > d$ ,
- the injection of  $d$  errors if and only if  $d(C) > d$ .

If  $D = C^\perp$ , then  $C$  and  $D$  are so-called *linear complementary dual (LCD)* codes.

The security parameter of an LCD code  $C$  when used in so-called *orthogonal direct sum masking (ODSM)* is then simply  $d(C) - 1$ .

The notion of LCD code is anterior to DSM, due to Yang and Massey.

We denote  $G'$  by  $H$  and

$$z = x \times G + y \times H \text{ implies :}$$

$$z \times H^t = y \times H \times H^t \text{ and } z \times G^t = x \times G \times G^t.$$

The matrices  $H \times H^t$  and  $G \times G^t$  are invertible.

Since the introduction of DSM, a hundred papers have proposed constructions.

- ▶ Robust codes, algebraic manipulation detection (AMD) codes and vectorial functions

In many cases of error detection, the assumption that the most probable errors have low Hamming weight cannot be guaranteed.

It is even often almost impossible to predict the error patterns.

This situation of unpredictability is similar to FIA where the error distribution within a device is controlled by an adversary.

A large enough minimum distance is then not efficient for a code.

**Definition 2.** A code  $C \subset \mathbb{F}_q^n$  (linear or not) is called  $R$ -robust if :

$$R_C = \max_{0 \neq e \in \mathbb{F}_q^n} |C \cap (e + C)| \leq R.$$

A binary  $R$ -robust code  $C$  of length  $n$  with  $M = |C|$  is denoted by a triple  $(n, M, R)$ .

The code can be *systematic*, i.e. we can have, up to permutation :

$$C = \{(x, F(x)); x \in \mathbb{F}_q^I\}.$$

This is more practical for error detection in computer hardware thanks to the separation between information bits and check bits.

The probability of error masking equals :

$$Q(e) = \frac{|C \cap (e + C)|}{|C|}. \quad (1)$$

The *worst error masking probability*  $\max_{e \neq 0} Q(e)$  equals then  $\frac{R_C}{|C|}$ .

A code is called *robust* if this value is strictly less than 1.

We have :

$$\max_{e \neq 0} Q(e) \geq \frac{|C| - 1}{q^n - 1}.$$

A code is called *uniformly robust* or *perfect robust* if there is equality, *i.e.*  $Q(e)$  is constant for  $e \neq 0$ , *i.e.*  $C$  is a *difference set* in  $(\mathbb{F}_q^n, +)$ .

If  $q = 2$ , the *indicator* function of  $C$  is bent.

Then  $d_C = 1$  and the code cannot be systematic.

**Proposition 3.** *(Kulikowski, Karpovsky, Taubin)*

*Let  $C = \{(x, F(x)), x \in \mathbb{F}_2^k\}$ , where  $F : \mathbb{F}_2^k \mapsto \mathbb{F}_2^r$ . The worst error masking probability of  $C$  equals the differential uniformity of  $F$  divided by  $2^k$ , and is then bounded below by  $2^{-r}$  and equals this optimum if and only if  $F$  is perfect nonlinear.*

Indeed, denoting  $e = (a, b)$ , we have :

$$\begin{aligned} |C \cap (e + C)| &= \left| \left\{ (x, y) \in (\mathbb{F}_2^k)^2; \left\{ \begin{array}{l} x = y + a \\ F(x) = F(y) + b \end{array} \right\} \right\} \right| \\ &= |(D_a F)^{-1}(b)|. \end{aligned}$$

The efficiency of these codes depends on the fact that the data be uniformly distributed.

This limitation can be overcome by (strong) algebraic manipulation detection (AMD) codes.

**Algebraic manipulation** : the attacker is able to modify the value of some abstract data storage device denoted by  $\Sigma(G)$ , without having read-access to the data.

The attacker is not able to obtain information about the element  $g$  stored in  $\Sigma(G)$ .

However, he can add an error  $e \in G$  of his choice.

This models the situation with *linear secret sharing schemes* with dishonest players, who can cause the reconstruction of a modified secret  $s' \neq s$ , and can control  $s - s'$ , thanks to the linearity of the secret sharing.

*Algebraic manipulation detection* (AMD) codes encode an original information  $s \in S$  as an element of  $g \in G$  in such way that any algebraic manipulation is detected with high probability.

No secret key is needed.



**Definition 4.** *An AMD code is a pair of two functions :*

- *a probabilistic encoding function  $E : S \rightarrow G$ ,*
- *a deterministic decoding function  $D : G \rightarrow S \cup \{\perp\}$ , where  $\perp \notin S$  symbolizes that algebraic manipulation has been detected, satisfying that  $D(E(s)) = s$  with probability 1 for every  $s \in S$ .*

*The AMD code is called  $\epsilon$ -secure for  $\epsilon > 0$  if, for every  $s \in S$  and for every  $e \in G$ , the probability that  $D(E(s) + e) \notin \{s, \perp\}$  is at most  $\epsilon$ .*

*A systematic AMD code is an AMD code in which set  $S$  is a group and the encoding function  $E$  has the form*

$$\begin{aligned} E : S &\rightarrow G = S \times G_1 \times G_2 \\ s &\rightarrow (s, x, F(x, s)). \end{aligned} \tag{2}$$

# Fully homomorphic encryption and related questions on Boolean functions

*Recent years :*

1. Proliferation of small embedded devices with limited computing facilities,
2. Apparition of cloud services with extensive storage and computing facilities.

The outsourcing of data processing raises new privacy concerns. Users want to prevent the server from learning about their data.

Gentry's Fully Homomorphic Encryption (FHE) scheme brings a perfect conceptual answer :

$$\mathbf{C}^H(m + m') = \mathbf{C}^H(m) + \mathbf{C}^H(m'); \quad \mathbf{C}^H(mm') = \mathbf{C}^H(m) \mathbf{C}^H(m').$$

If Alice wants to compute  $f(m)$ , she can send  $\mathbf{C}^H(m)$  to Claude, who computes  $f(\mathbf{C}^H(m)) = \mathbf{C}^H(f(m))$ .

After decryption, Alice gets  $f(m)$ , but Claude has not learned anything about  $m$  nor about  $f(m)$  (but he knows  $f$ ).

But in practice,  $\mathbf{C}^H(m)$  is too large for Alice.

Alice needs then to use a hybrid Symmetric Encryption-FHE protocol.

## *Typical Framework :*

1. *Initialization.* Alice sends to Claude :
  - her homomorphic public key  $pk^H$ ,
  - the homomorphic ciphertext of her symmetric key  $C^H(sk^S)$ .
2. *Storage.* Alice encrypts her data  $m$  with the symmetric encryption scheme  $C^S$ , and sends  $C^S(m)$  to Claude.
3. *Evaluation.* Claude calculates  $C^H(C^S(m))$  and homomorphically evaluates the decryption of the symmetric scheme on Alice's data and gets  $C^H(m)$ .
4. *Computation.* Claude homomorphically executes the treatment  $f$  on Alice's data, and gets  $C^H(f(m))$ .

5. *Result.* Claude sends  $\mathbf{C}^H(f(m))$  and Alice gets  $f(m)$ .

*Bottleneck :*

2nd and 3rd generations of FHE are noise-based (LWE) and need expensive “bootstrapping” when the noise grows too much.

The choice of the symmetric cipher  $\mathbf{C}^S$  is central for reducing cost.

The multiplicative depth of AES being too large, other symmetric encryption schemes have been investigated :

- a stream cipher : *Kreyvium* (FSE 2016),
- block ciphers *LowMC* (EUROCRYPT 2015), *Rasta* (CRYPTO 2018).

► The FLIP cryptosystem (EuroCrypt 2016)

The stream cipher FLIP (Méaux, Journault, Standaert, C.C., EURO-CRYPT 2016) is based on a cipher model called the *filter permutator*.

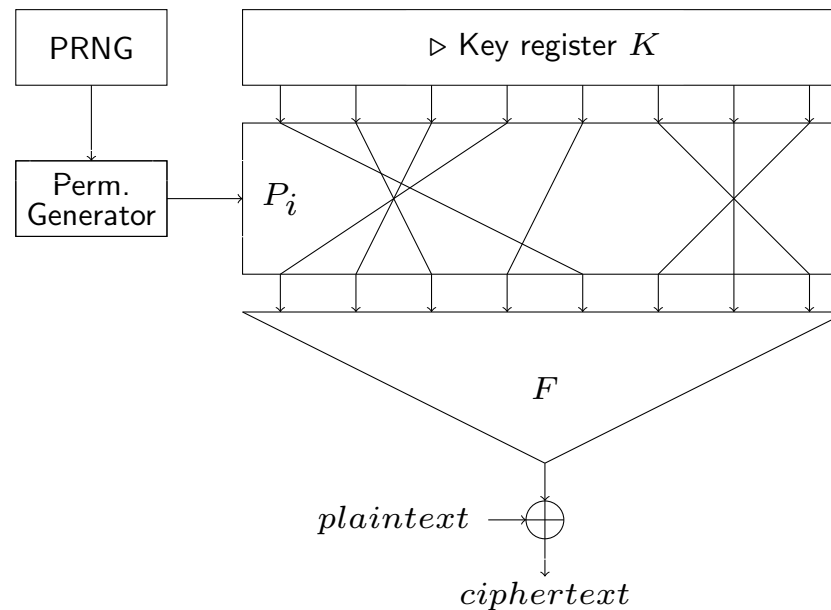


FIGURE 1: Filter permutator construction.

Function  $F$  has  $N = n_1 + n_2 + n_3 \geq 500$  variables, where  $n_2$  is even and  $n_3 = \frac{k(k+1)}{2}t$ . It is defined as :

$$F(x_0, \dots, x_{n_1-1}, y_0, \dots, y_{n_2-1}, z_0, \dots, z_{n_3-1}) =$$

$$\sum_{i=0}^{n_1-1} x_i + \sum_{i=0}^{n_2/2-1} y_{2i} y_{2i+1} +$$

$$\sum_{j=1}^t T_k \left( z_{\frac{(j-1)k(k+1)}{2}}, z_{\frac{(j-1)k(k+1)}{2}+1}, \dots, z_{\frac{(j-1)k(k+1)}{2} + \frac{k(k+1)}{2} - 1} \right),$$

where the so-called *triangular* function  $T_k$  is defined as :

$$T_k(z_0, \dots, z_{j-1}) = z_0 + z_1 z_2 + z_3 z_4 z_5 + \dots + z_{\frac{k(k-1)}{2}} \dots z_{\frac{k(k+1)}{2} - 1}.$$

FLIP : 4 filtering functions proposed :

Name	$N$	$n_1$	$n_2$	$t$	$k$	$\lambda$
FLIP-530	530	42	128	8	9	80
FLIP-662	662	46	136	4	15	80
FLIP-1394	1394	82	224	8	16	128
FLIP-1704	1704	86	238	5	23	128

TABLE 1:  $N$  : total number of variables,  $n_1$  : linear part,  $n_2$  : quadratic part,  $t$  : number of triangular functions,  $k$  : degree of the triangular functions;  $\lambda$  : resulting security parameter.



There exists a Guess and Determine attack on a preliminary version of FLIP, by Sébastien Duval, Virginie Lallemand and Yann Rotella (CRYPTO 2015).

It is not efficient on the regular version of FLIP.

But, by definition, in the filter permutator, the input to  $F$  has *constant Hamming weight* (equal to the weight of the secret key).

The study of Boolean functions on such restricted sets of inputs have been made (C.C., P. Méaux, Y. Rotella, S. Mesnager et al.).

# Local pseudorandom generators and related criteria on Boolean functions

*Principle* : allow expanding short random strings (like private keys), called seeds, into pseudorandom strings, whose length is significantly larger, say,  $\mathcal{O}(n^s)$  where  $n$  is the length of the seed.

Called *local* if each output bit depends on a constant number  $d$  of input bits.

Only known example : Goldreich's PRG, which applies a simple  $d$ -variable Boolean function (Goldreich calls it a  $d$ -ary predicate) to public random subsets of size  $d$  of the seed.

Let  $(S_1, \dots, S_m)$  be a list of  $m$  subsets of  $\{1, \dots, n\}$  of size  $d$ , and let  $f$  be a Boolean function in  $d$  variables (the so-called predicate).

The corresponding *Goldreich's function*  $G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$  is defined as

$$G(x) = f(S_1(x)), f(S_2(x)), \dots, f(S_m(x))$$

for every  $x \in \mathbb{F}_2^n$ , where  $S_i(x)$  is a vector made of those bits of  $x$  indexed by  $S_i$ .

- To avoid an attack by Gaussian elimination, the predicate  $f$  must not be linear.
- The higher the algebraic degree, the better (a predicate of algebraic degree  $s$  cannot be pseudorandom for a stretch  $s$ ).

- The predicate must be such that, when fixing some number  $r$  of input bits to  $f$ , its algebraic degree remains large.
- The algebraic immunity  $AI(f)$  plays also a direct role and should be large enough (larger than  $s$ ).
- There is also an attack when the output to the function is correlated with a number of its input bits smaller than or equal to  $\frac{s}{2}$ , and  $f$  should then be *resilient* with a sufficient order. At least 2-resilient.

Example of a 5-variable function :  $f(x) = x_1 \oplus x_2 \oplus x_3 \oplus x_4x_5$ .

A general structure has been proposed for predicates :  
the direct sum of  $\bigoplus_{i=1}^k x_i$  and of the *majority function* in  $n - k$   
variables.

No attack is known on such functions when  $k \geq 2s$  and  $\lceil \frac{n-k}{2} \rceil \geq s$ .

Open question by Applebaum and Lovett : given  $e$  and  $k$ , what is  
the smallest number of variables of a Boolean function of algebraic  
immunity at least  $e$  and of resiliency order at least  $k$  ?

## The Gowers norm on pseudo-Boolean functions

The Gowers uniformity norm has been introduced in 2001 in relation with arithmetic progressions in partitions of  $\{1, 2, \dots, M\}$ .

Intensively studied (by several Field medal winners) since 2001 and applied in additive combinatorics and in the probabilistic testing of specific properties of Boolean functions (knowing only a few of their values).

**Definition 5.** Let  $k, n$  be positive integers such that  $k < n$ . Let  $\varphi : \mathbb{F}_2^n \mapsto \mathbb{R}$ . The  $k$ th-order Gowers uniformity norm of  $\varphi$  equals :

$$\|\varphi\|_{U_k} = \left( \mathbb{E}_{x, x_1, \dots, x_k \in \mathbb{F}_2^n} \left[ \prod_{S \subseteq \{1, \dots, k\}} \varphi \left( x + \sum_{i \in S} x_i \right) \right] \right)^{\frac{1}{2^k}},$$

where  $\mathbb{E}_{x, x_1, \dots, x_k \in \mathbb{F}_2^n}$  is the notation for arithmetic mean (i.e. for expectation in uniform probability).

When  $\varphi$  is the sign function of a Boolean function  $f$ , this results in a measure related to the higher order nonlinearity.

For every  $\varphi$ , the sequence  $(\|\varphi\|_{U_k})_{k \geq 1}$  is non-decreasing :

$$\|\varphi\|_{U_1} \leq \|\varphi\|_{U_2} \leq \cdots \leq \|\varphi\|_{U_k} \leq \cdots$$

For every  $k \geq 2$ ,  $\|\cdot\|_{U_k}$  is a norm :

$$\|\varphi\|_{U_k} = 0 \text{ iff } \varphi = 0 \text{ and } \|\varphi + \psi\|_{U_k} \leq \|\varphi\|_{U_k} + \|\psi\|_{U_k}.$$

For  $\varphi = f_\chi = (-1)^f$ ,  $\|f_\chi\|_{U_k}$  equals the  $2^k$ -th root of the average value of  $2^{-n} \mathcal{F}(D_{a_1} D_{a_2} \cdots D_{a_k} f)$ , where  $\mathcal{F}(g) = \sum_{x \in \mathbb{F}_2^n} (-1)^{g(x)}$ , when  $a_1, a_2, \dots, a_k$  range independently over  $\mathbb{F}_2^n$ .

We have  $\|f_\chi\|_{U_k} \leq 1$ , with equality if and only if  $f$  has algebraic degree at most  $k - 1$ .



$\|f_\chi\|_{U_2}$  is related to the second moment  $\mathcal{V}(f)$  of the autocorrelation coefficients by :

$$(\|f_\chi\|_{U_2})^4 = 2^{-3n} \mathcal{V}(f). \quad (3)$$

We have :

$$nl(f) \leq 2^{n-1} - 2^{n-1}(\|f_\chi\|_{U_2})^2 \leq 2^{n-1} - 2^{\frac{3n}{4}-1}\|f_\chi\|_{U_2},$$

and these two inequalities are equalities if and only if  $f$  is bent.

We have also :

$$\|f_\chi\|_{U_2} = 2^{-n} \left( \sum_{b \in \mathbb{F}_2^n} W_f^4(b) \right)^{\frac{1}{4}}, \quad (4)$$

that is, up to a multiplicative coefficient,  $\|f_\chi\|_{U_2}$  equals the quartic mean of the Walsh transform of  $f$ .