KU LEUVEN

COSIC

Threshold Implementations

Svetla Nikova

Threshold Implementations

- A provably secure countermeasure
- Against (first) order power analysis

based on

multi party computation and *secret sharing*

Outline

- Threshold Implementations (update)
- Applications of TI
- Higher-order TI

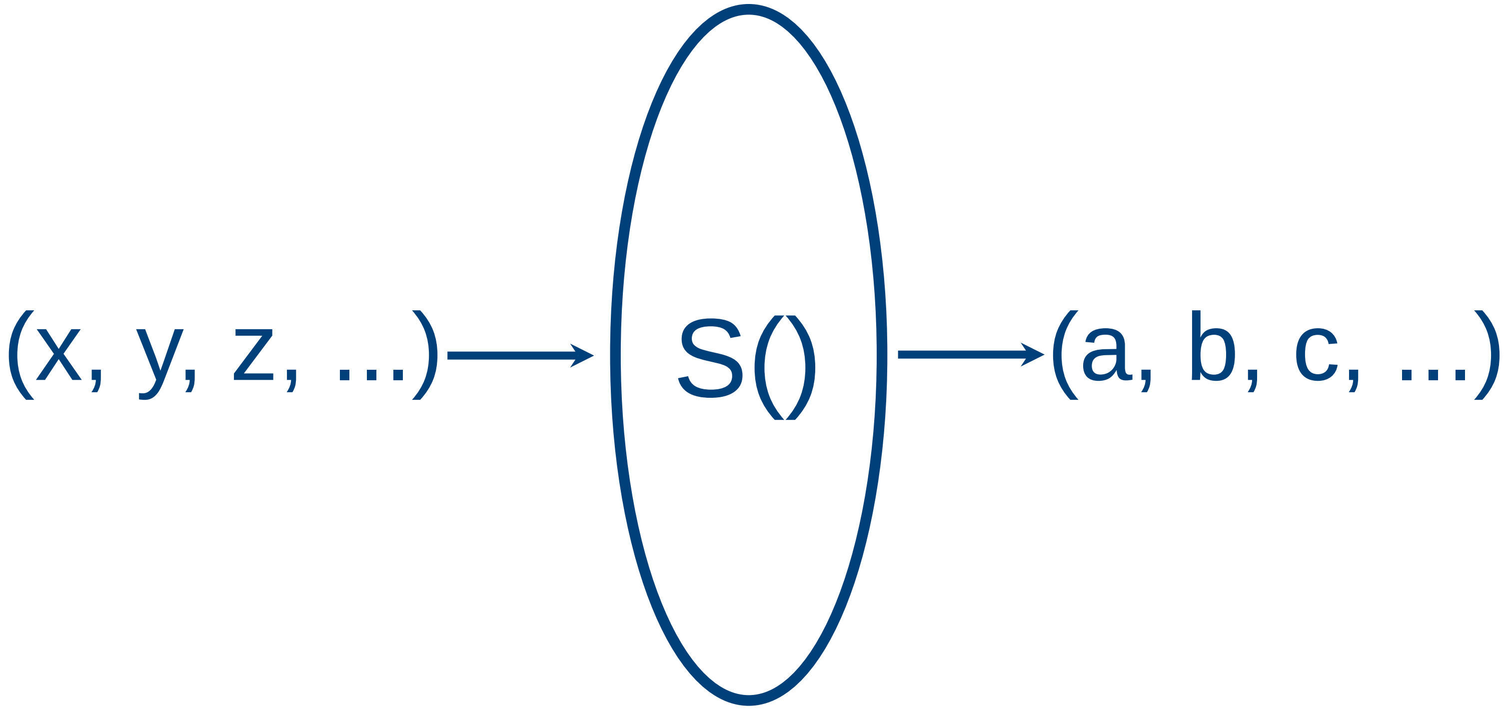
Countermeasures

- Hardware countermeasures
  - Balancing power consumption [Tiri et al., CHES'03]

- Masking
  - Randomizing intermediate values [Chari et al., Crypto'99; Goubin et al., CHES'99]
  - Threshold Implementations [Nikova et al., ICICS'06]
  - Shamir's Secret Sharing [Goubin et al,. Prouff et al., CHES'11]

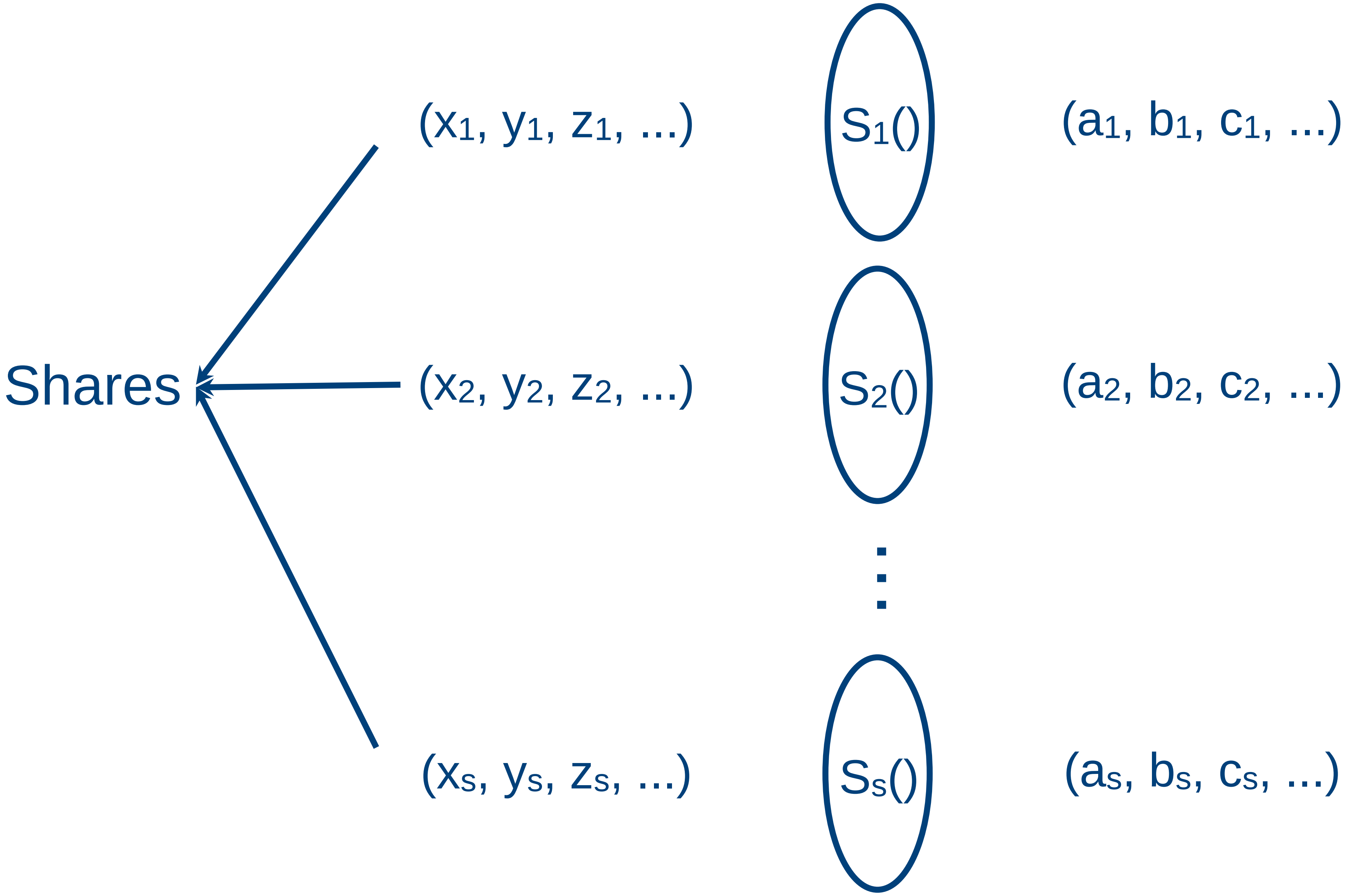
- Leakage-Resilient Crypto

Threshold Implementations



“Threshold Implementations … ”,
S.Nikova, V.Rijmen et al. 2006, 2008, 2010 (JoC).

Threshold Implementations



Threshold Implementations

$(x_1, y_1, z_1, \ldots)$ $\quad S_1()$ $\quad (a_1, b_1, c_1, \ldots)$

$\oplus$ $\quad\quad\quad\quad \oplus$

$(x_2, y_2, z_2, \ldots)$ $\quad S_2()$ $\quad (a_2, b_2, c_2, \ldots)$

$\oplus$ $\quad\quad\quad\quad \oplus$

$\vdots$ $\quad \vdots$ $\quad \vdots$

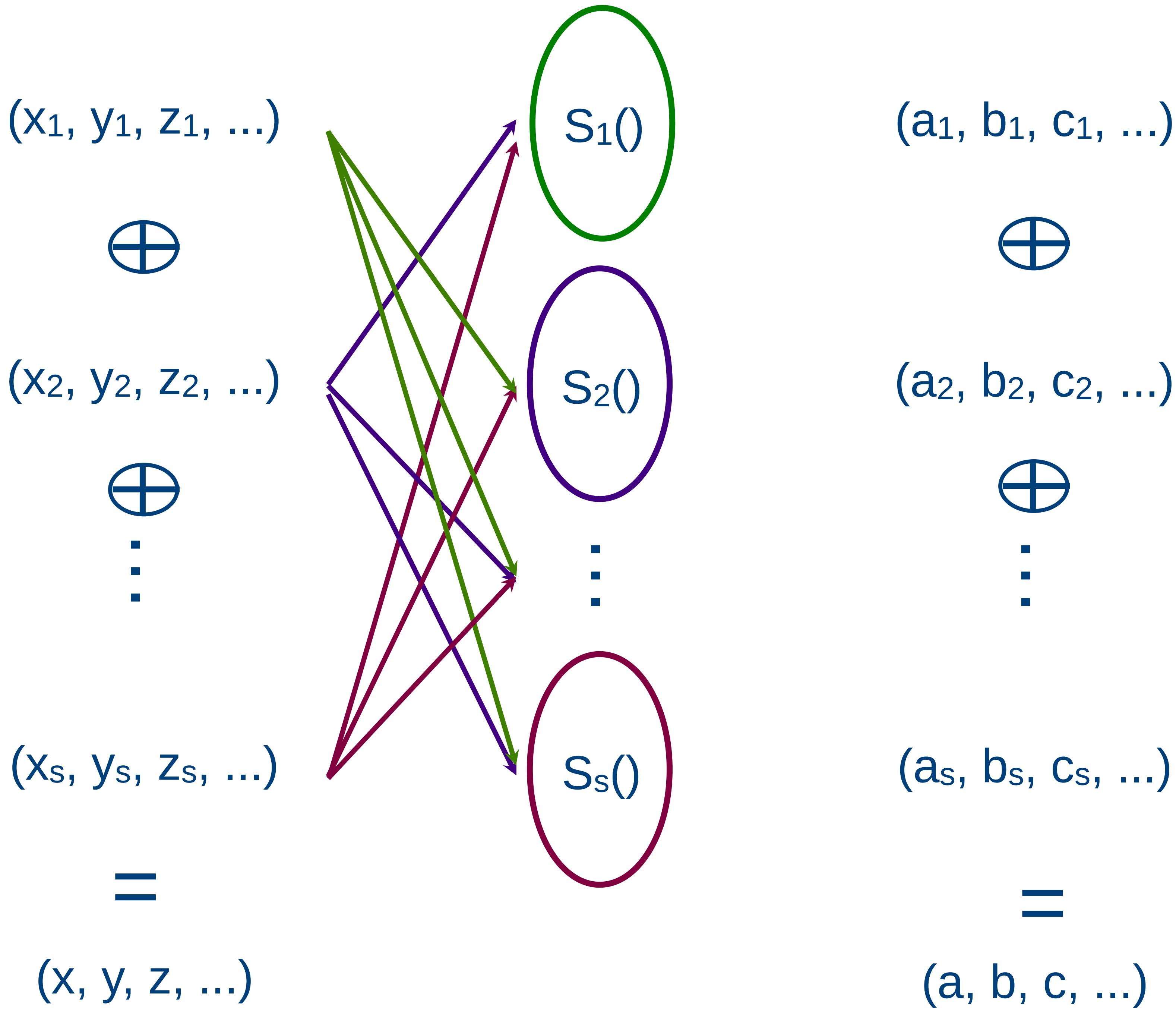
$(x_s, y_s, z_s, \ldots)$ $\quad S_s()$ $\quad (a_s, b_s, c_s, \ldots)$

$=$ $\quad\quad\quad\quad =$

$(x, y, z, \ldots)$ $\quad\quad\quad\quad (a, b, c, \ldots)$

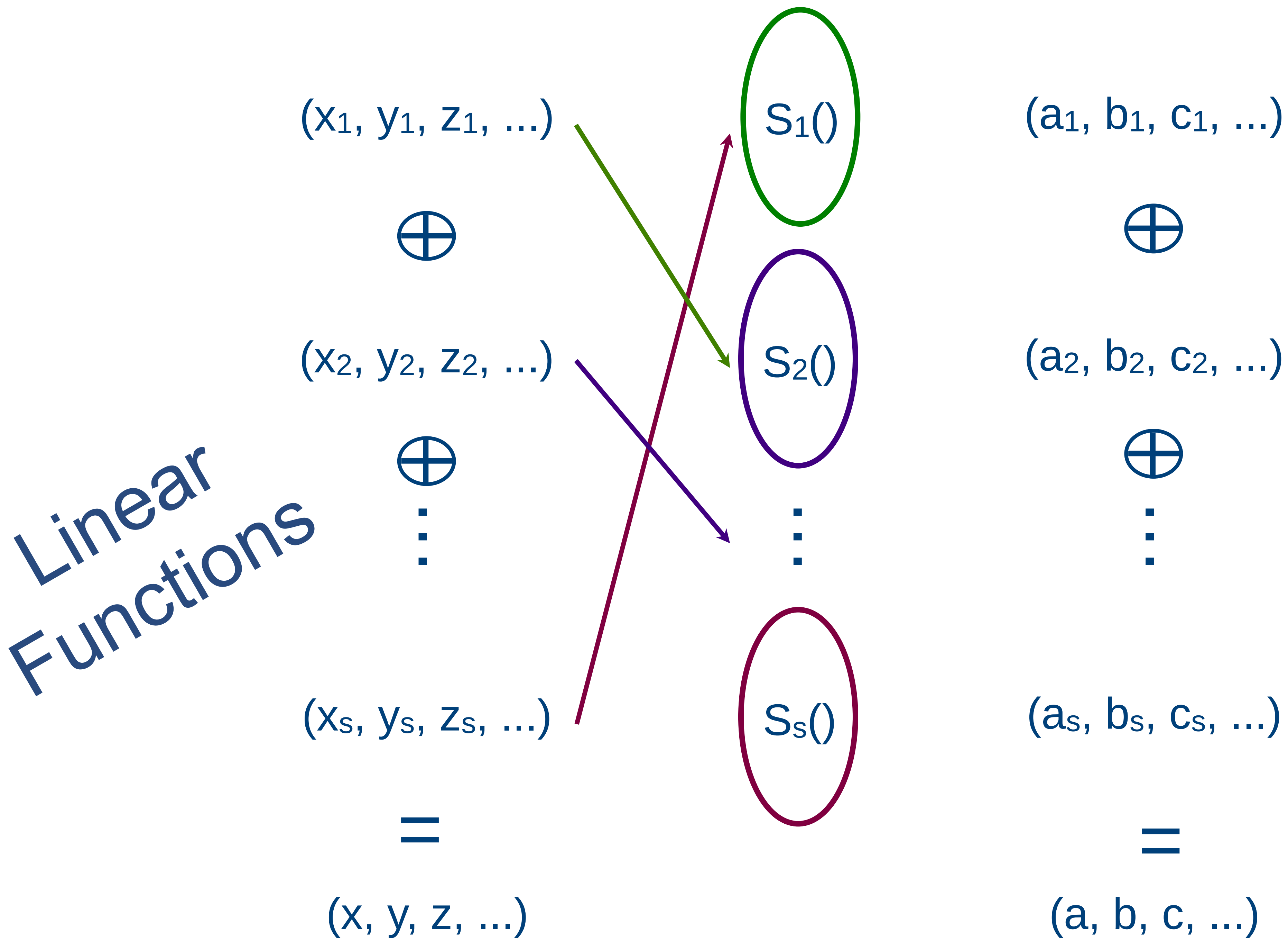
Correct

Threshold Implementations



Correct, Non-complete

Threshold Implementations



Correct, Non-complete

Threshold Implementations

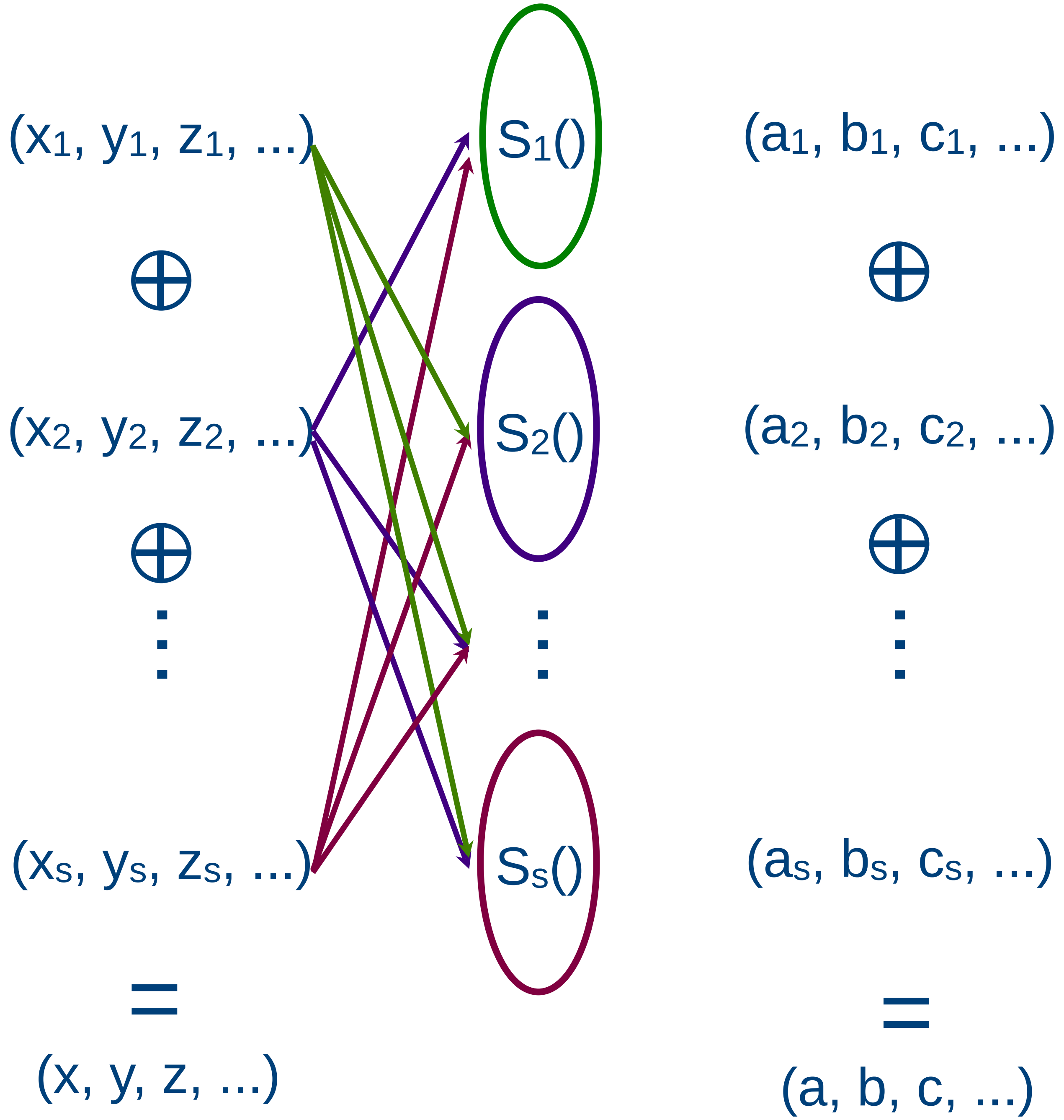
Non-completeness

$$S(x, y, z) = x + yz$$

$$\begin{aligned} S_1 &= x_2 + y_2 z_2 + y_2 z_3 + y_3 z_2 \\ S_2 &= x_3 + y_3 z_3 + y_3 z_1 + y_1 z_3 \\ S_3 &= x_1 + y_1 z_1 + y_1 z_2 + y_2 z_1 \end{aligned}$$

To protect a function with degree *d*, at least *d+1* shares are required

Threshold Implementations



Correct, Non-complete, Uniform

Threshold Implementations

Uniformity

a	b	f = a AND b
0	0	0
0	1	0
1	0	0
1	1	1

f	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
0	4	0	0	4	0	4	4	0
0	4	0	0	4	0	4	4	0
0	4	0	0	4	0	4	4	0
1	0	4	4	0	4	0	0	4
0	12	0	0	12	0	12	12	0
1	0	4	4	0	4	0	0	4

Threshold Implementations

Uniformity

A masking X is *uniform* if and only if there exists a constant p such that for all x we have:

$$\text{if } X \in \mathrm{Sh}(x) \text{ then } \Pr(X|x) = p, \text{ else } \Pr(X|x) = 0\,.$$

If unshared function is a permutation,
the shared function should also be a permutation

Threshold Implementations

If the masking of x is uniform, then the stochastic functions S_i and x are independent (for any choice of i).

$$\Downarrow$$

If the masking of x is uniform and the circuit S is non-complete, then any single component function of S does not leak information on x.

No leak even in the presence of glitches!

Threshold Implementations

Uniformity

f	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
0	4	0	0	4	0	4	4	0
0	4	0	0	4	0	4	4	0
0	4	0	0	4	0	4	4	0
1	0	4	4	0	4	0	0	4
0	12	0	0	12	0	12	12	0
1	0	4	4	0	4	0	0	4

(a,b,d)	000	011	101	110	001	010	100	111
(0,0,0)	37	9	9	9	0	0	0	0
(0,0,1)	37	9	9	9	0	0	0	0
(0,1,0)	37	9	9	9	0	0	0	0
(0,1,1)	37	9	9	9	0	0	0	0
(1,0,0)	37	9	9	9	0	0	0	0
(1,0,1)	37	9	9	9	0	0	0	0
(1,1,0)	31	11	11	11	0	0	0	0
(1,1,1)	0	0	0	0	21	21	21	1

Threshold Implementations
Uniformity and a remedy

- Firstly, we can apply re-masking, i.e. by adding new masks to the shares we make the distribution uniform.
- Secondly, we can impose an extra condition on F, such that the distribution of the output is always uniform.

Definition

The circuit F is *uniform* if and only if

$$\forall x \in \mathcal{F}^m, \forall y \in \mathcal{F}^n \text{ with } f(x) = y, \forall Y \in \mathrm{Sh}(y) :$$

$$|\{X \in \mathrm{Sh}(x) | F(X) = Y\}| = \frac{2^{m(s_x - 1)}}{2^{n(s_y - 1)}}$$

- If X, the masking of x is uniform and the circuit F is uniform, then the masking Y = F(X) of y = f (x) is uniform.

Threshold Implementations

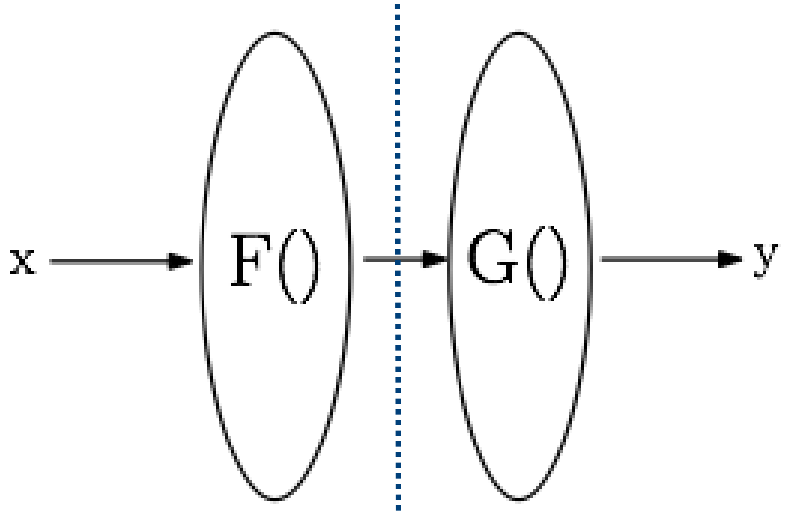
Observations

- ✓ Linear functions are easy to protect
- As the nonlinearity increases
  - X DPA becomes easier
  - X Sharing becomes costly
  - ✓ S-boxes become mathematically stronger

Decomposing nonlinear functions

Threshold Implementations

Decomposing nonlinear functions



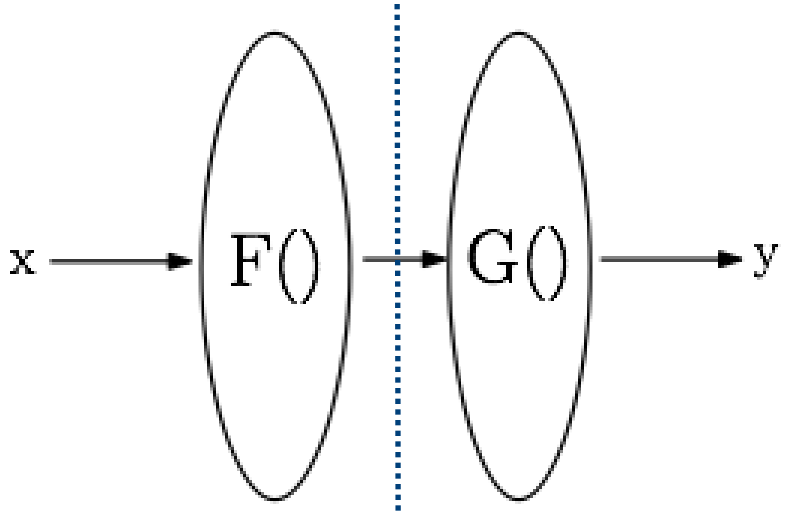
S = G o F

Most of the block ciphers use 4x4 permutations

4x4 permutations have at most degree 3

Threshold Implementations

Decomposing nonlinear functions



S = G o F

All nxn affine bijections are in alternating group A_{2^n}

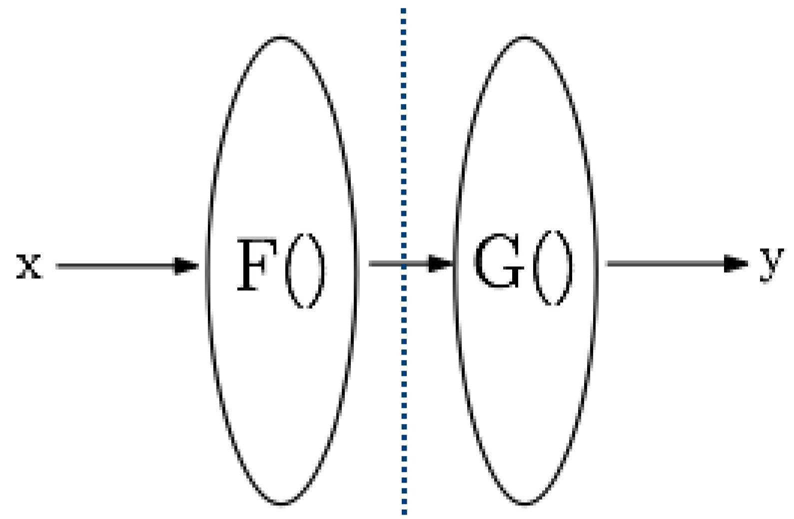
All 4x4 quadratic S-boxes belong to A_{16}

A 4x4 bijection can be decomposed using quadratic bijections IFF it belongs to A_{16}

KU LEUVEN

Threshold Implementations

Decomposing nonlinear functions



S = G o F

302 affine equivalent classes of 4x4 S-boxes

S'=AoSoB

half of the 4x4 S-boxes belong to A_{16} ➡ 3 shares

Threshold Implementations

Decomposing nonlinear functions

remark	unshared	3 shares				4 shares			5 shares
		1	2	3	4	1	2	3	1
affine	1	1				1			1
quadratic	6	5	1			6			6
cubic in A_{16}	30		28	2			30		30
cubic in A_{16}	114			113	1			114	114
cubic in $S_{16} \setminus A_{16}$	151					4	22	125	151

"Threshold Implementations of All 3 ×3 and 4 ×4 S-Boxes", B.Bilgin et al., CHES 2012.

Threshold Implementations

Decomposing nonlinear functions

remark	unshared	3 shares				4 shares			5 shares
		1	2	3	4	1	2	3	1
affine	1	1				1			1
quadratic	6	5	1			6			6
cubic in A_{16}	30		28	2			30		30
cubic in A_{16}	114			113	1			114	114
cubic in $S_{16} \setminus A_{16}$	151					4	22	125	151

Uniformity problem

Threshold Implementations

Decomposing nonlinear functions

remark	unshared	3 shares				4 shares			5 shares
		1	2	3	4	1	2	3	1
affine	1	1				1			1
quadratic	6	5	1			6			6
cubic in A_{16}	30		28	2			30		30
cubic in A_{16}	114			113	1			114	114
cubic in $S_{16} \setminus A_{16}$	151					4	22	125	151

Many S-boxes with good cryptographic properties

Threshold Implementations

Decomposing nonlinear functions

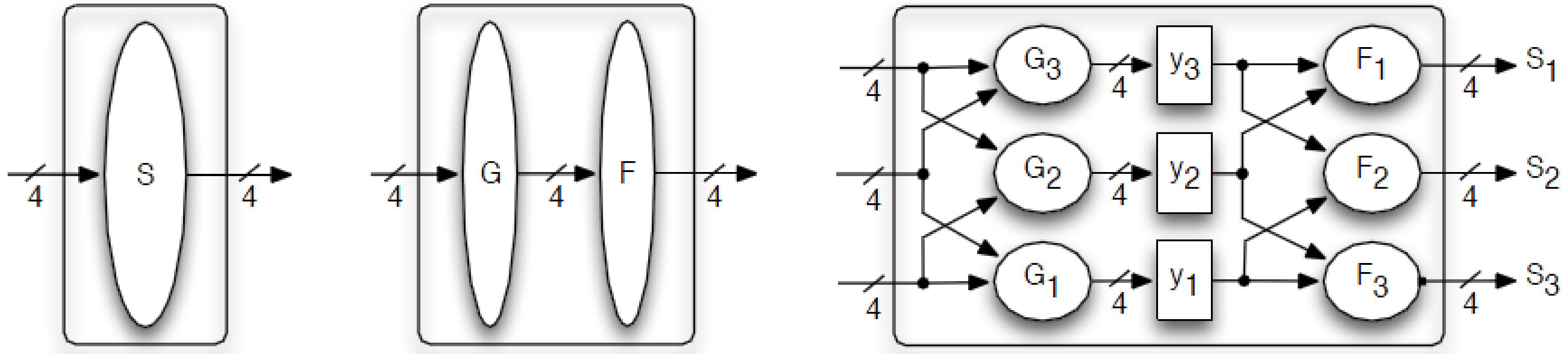
remark	unshared	3 shares				4 shares			5 shares
		1	2	3	4	1	2	3	1
affine	1	1				1			1
quadratic	6	5	1			6			6
cubic in A_{16}	30		28	2			30		30
cubic in A_{16}	114			113	1			114	114
cubic in $S_{16} \setminus A_{16}$	151					4	22	125	151

http://homes.esat.kuleuven.be/~snikova/ti_tools.html

Outline

- Threshold Implementations (update)
- Applications of TI
- Higher-order TI

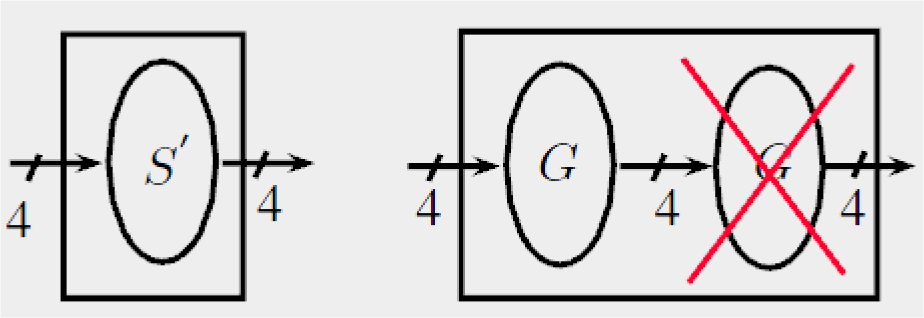
Applications - Present

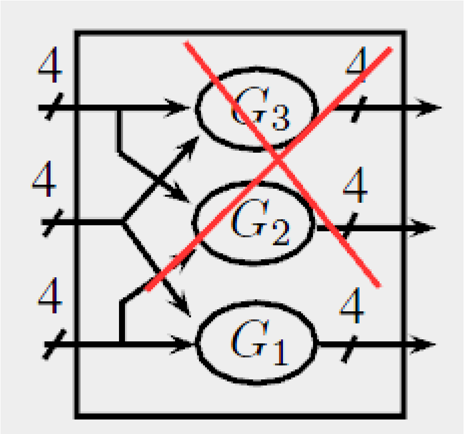


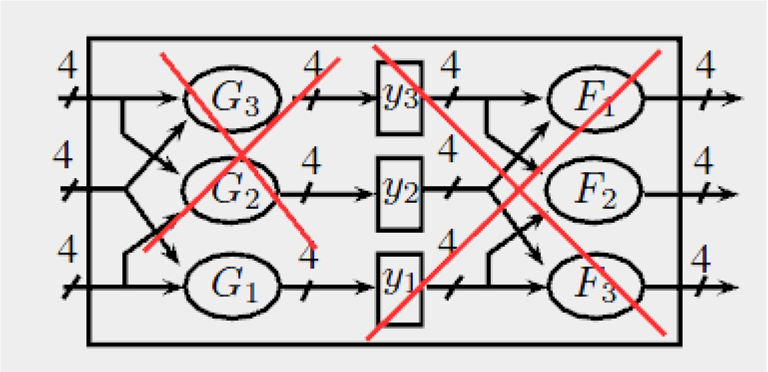
- "Side-Channel Resistant Crypto for less than 2300 GE", A.Poschmann et al., JOC 2010.
  - uses 4x4 S-box with degree 3
  - Implemented with 3 shares
  - 3,3 kGE (1,1 kGE unprotected)
  - 31×(16+1)+20 = 547 cycles

KU LEUVEN

Applications - Present

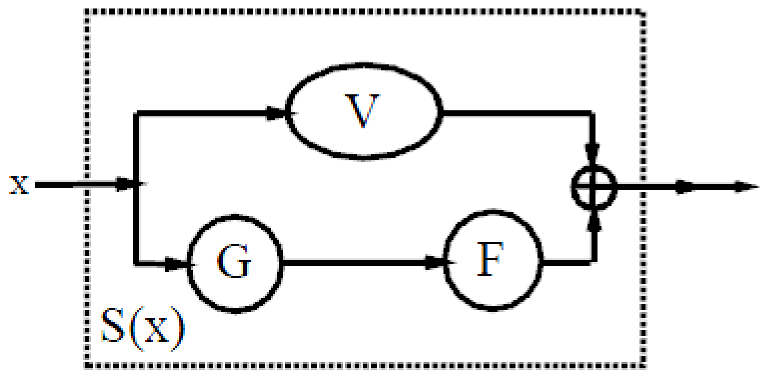


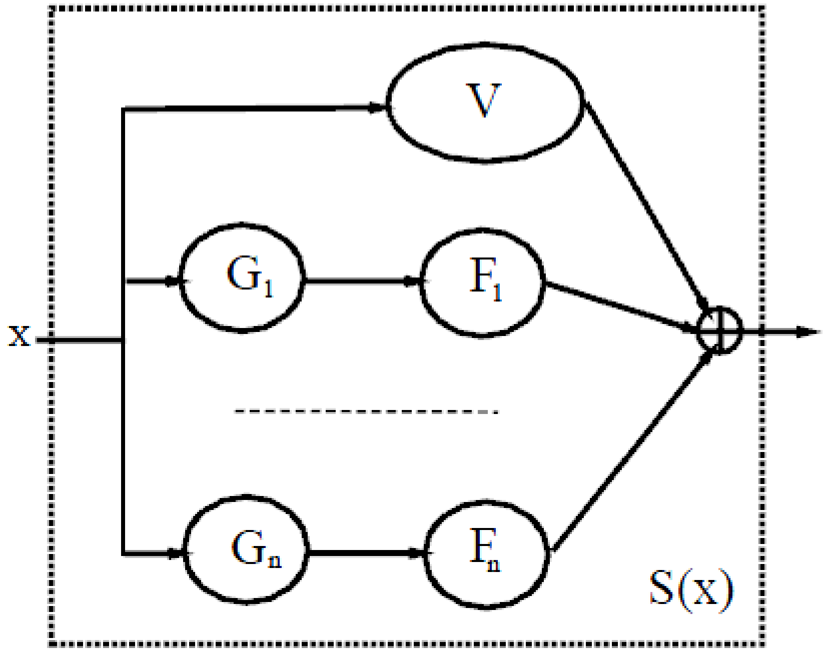




- "On 3-share Threshold Implementations for 4-bit S-boxes", S.Kutzner et al., COSADE 2013.
  - Implemented with 3 shares S` = G(G(.))
  - $G_1 = G_2 = G_3$
  - 3,0 kGE (-200 GE S-box)
  - 31×(16×**6**) + 20 = 2996 cycles

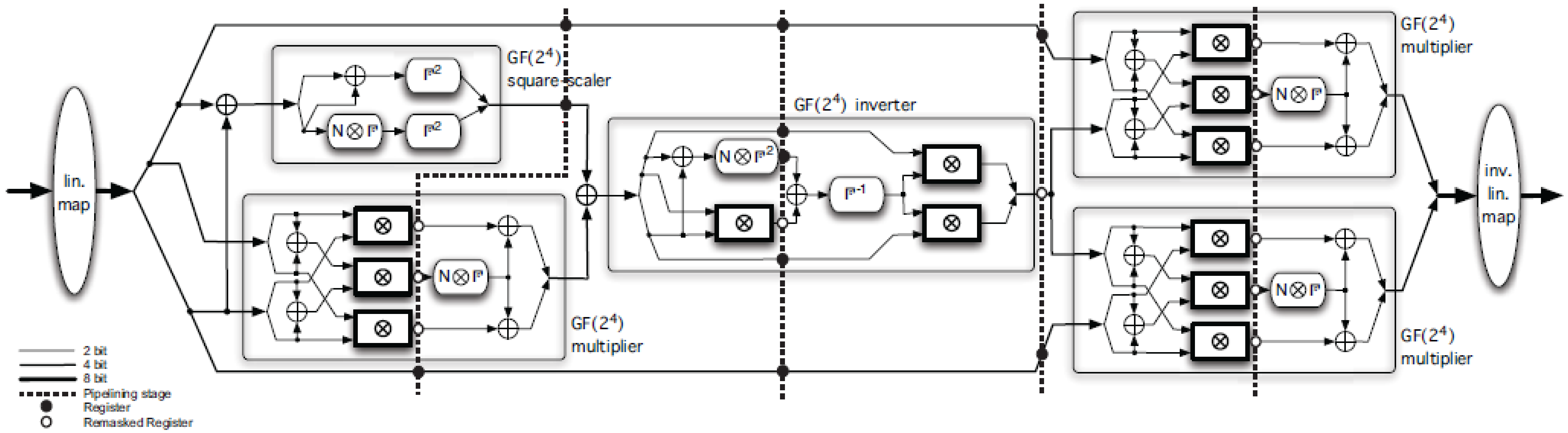
Applications





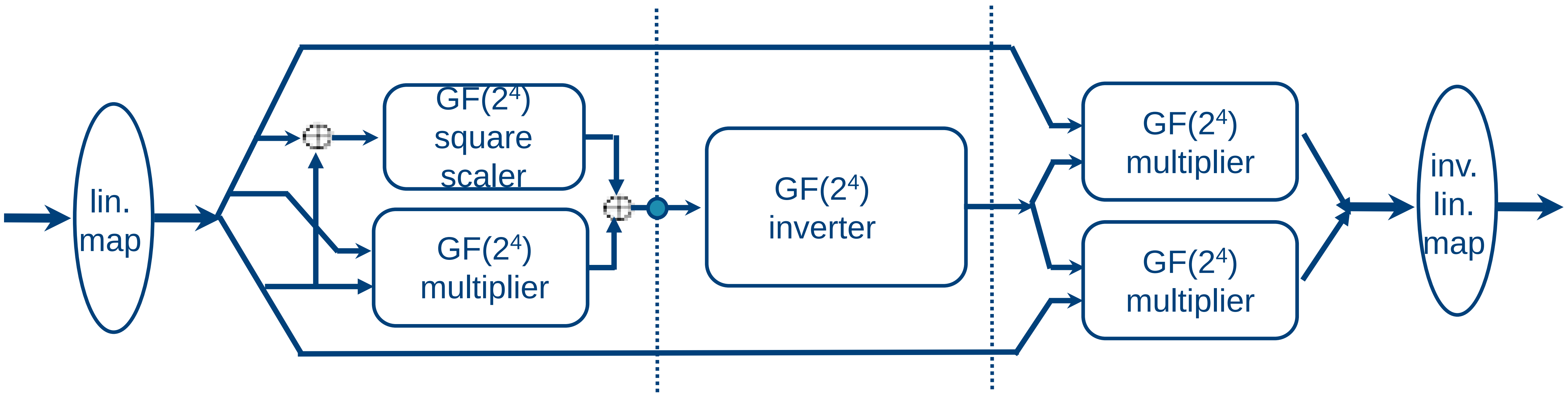
- "Enabling 3-share Threshold Implementations for any 4-bit S-box", S.Kutzner et al., ePrint Archive 2012.
  - Factorization S(.) = U(.) + V(.)
  - U(.) contains all the cubic terms, V(.) quadratic
  - U(.) = F(G(.)) with quadratic F(.) and G(.)

Applications - AES



- "Pushing the Limits: A Very Compact and a Threshold Implementation of AES", A.Moradi et al., Eurocrypt 2011.
  - uses 8x8 S-box with degree 7; 3 shares
  - Tower field approach down to GF(4); re-sharing (48 random bits per S-box)
  - 11.1 kGE (2,4 kGE unprotected)
  - 266 cycles (226 unprotected)

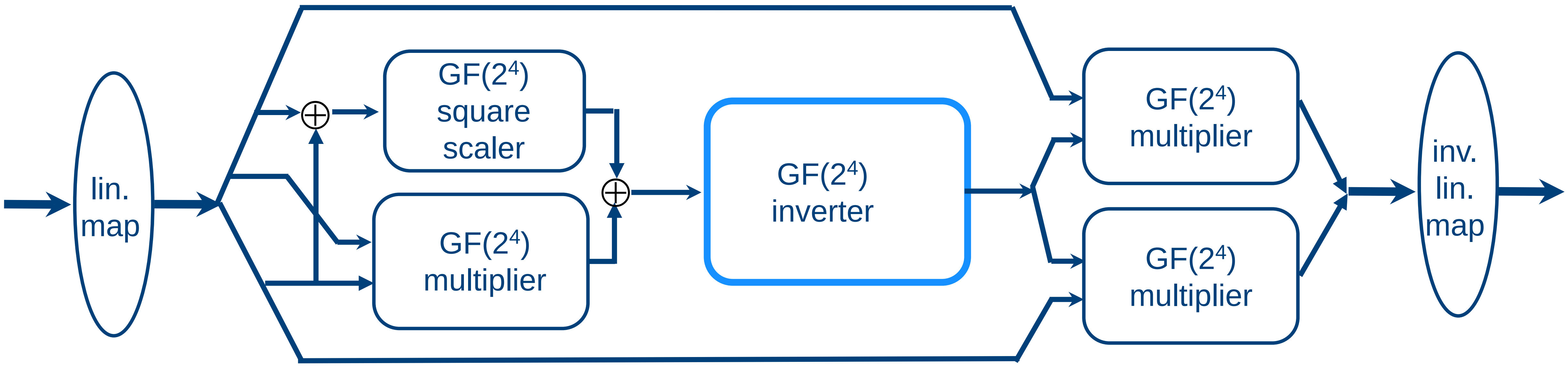
Applications - AES



- "A More Efficient AES Threshold Implementation", B.Bilgin et al., Africacrypt 2014.
  - Implemented with n shares
  - Tower field approach down to GF(16); re-sharing (44 random bits per S-box)
  - 8,2 kGE (-2,9 kGE)
  - 246 cycles (-20 cycles)

TI on AES

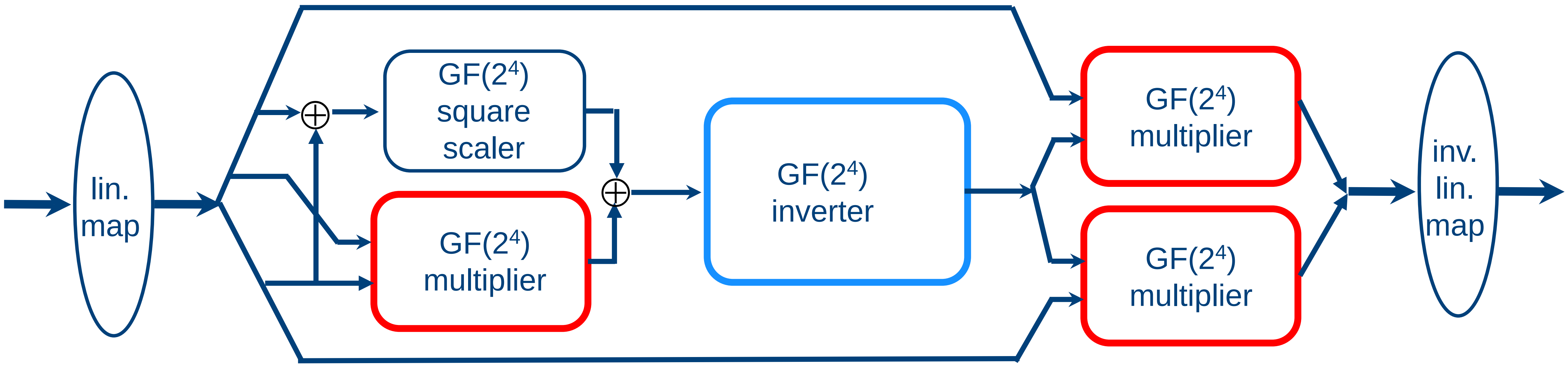
S-box



5 shares

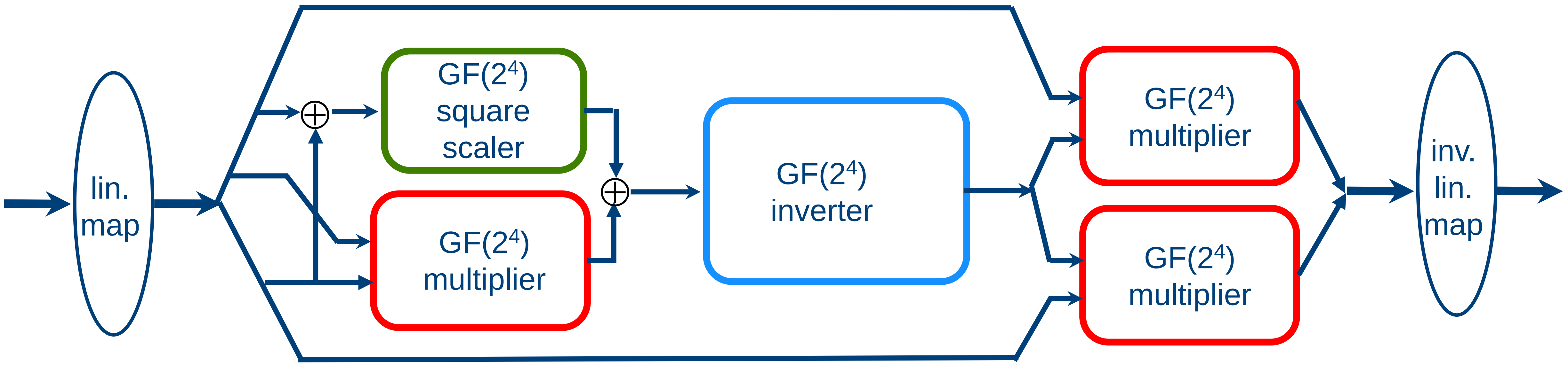
TI on AES

S-box



5 shares, 4 input 3 output shares

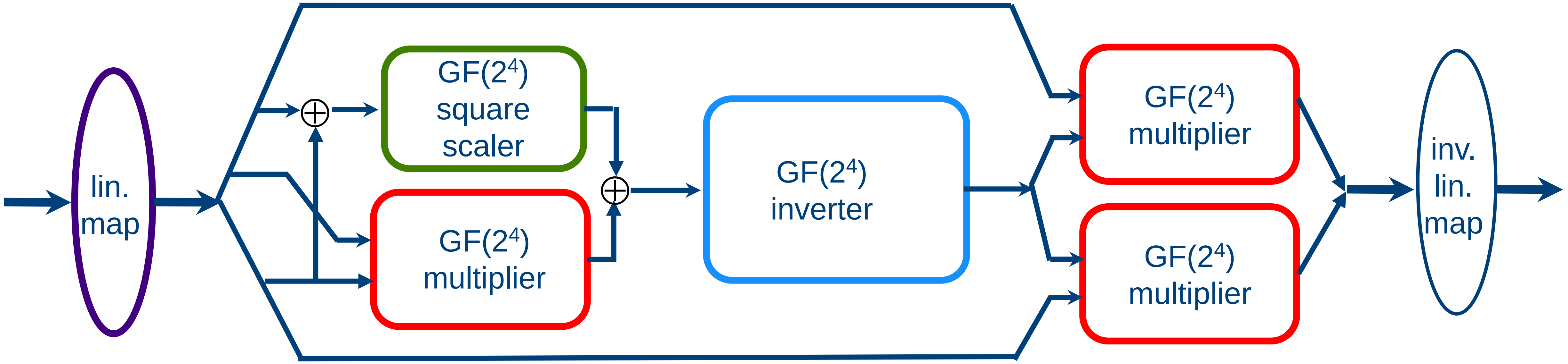
TI on AES
S-box



5 shares, 4 input 3 output shares, 2 shares

TI on AES

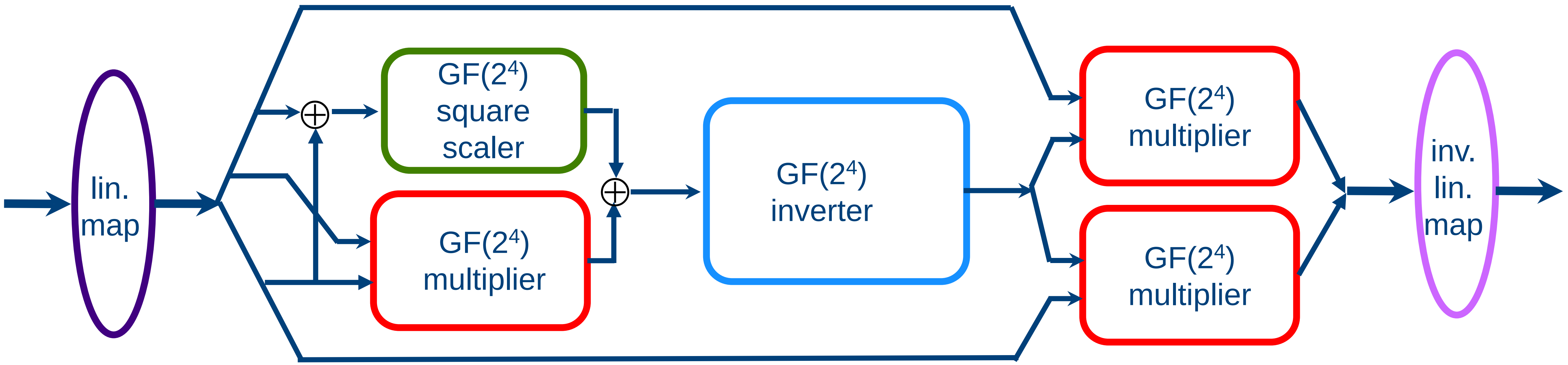
S-box



5 shares, 4 input 3 output shares, 2 shares, 4 shares

TI on AES

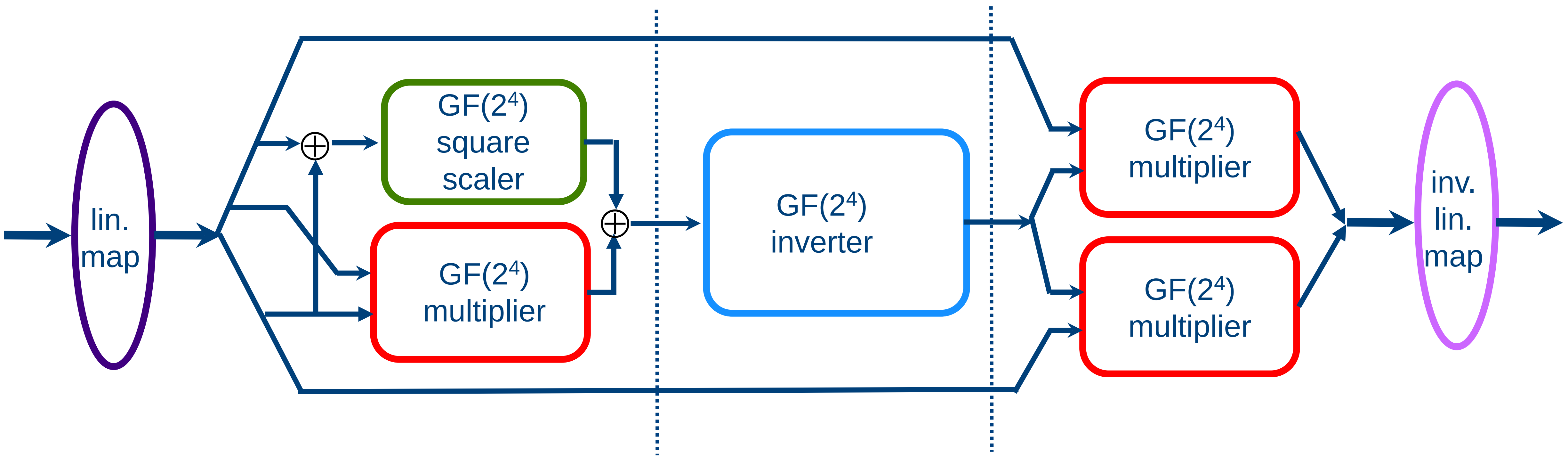
S-box



5 shares, 4 input 3 output shares, 2 shares, 4 shares, 3 shares

TI on AES

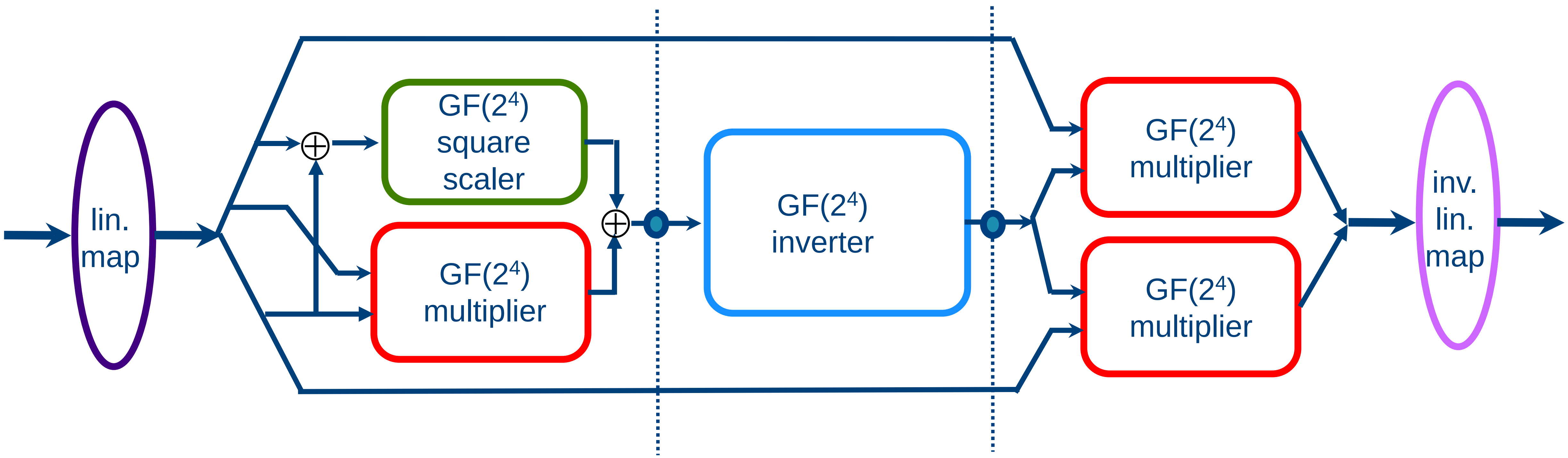
S-box



5 shares, 4 input 3 output shares, 2 shares, 4 shares, 3 shares

registers after every nonlinear function

TI on AES
S-box



5 shares, 4 input 3 output shares, 2 shares, 4 shares, 3 shares

registers after every nonlinear function

re-masking to change the number of shares

TI on AES
Implementation Results

	State Array	Key Array	S-box	Mix Col.	Cont.	MUXes	Other	Total	cycles	rand bits **
Moradi et al.	2529	2526	4244	1120	166	376	153	11114/11031	266	48
This paper	1698	1890	3708	770	221	746	69	9102	246	44
This paper*	1698	1890	3003	544	221	746	69	8171	246	44

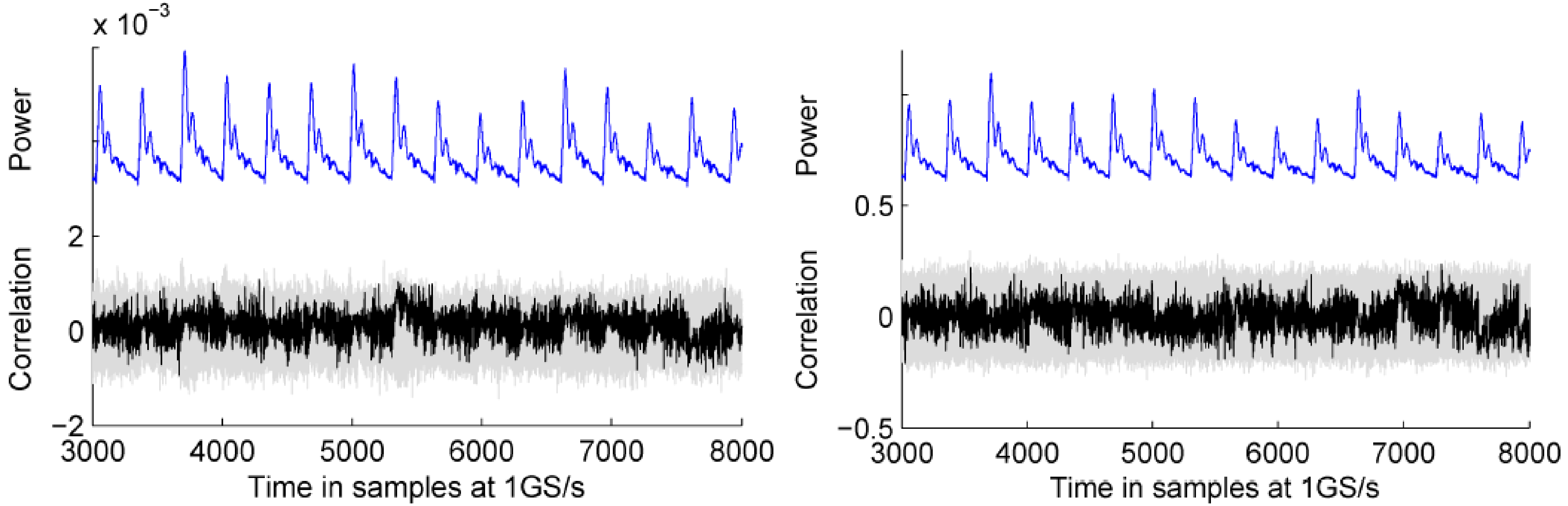
* compile_ultra ** per S-box

- Based on plain Canright S-box (233 GE)
- Based on plain Moradi et al.'s AES (2.4 GE)
- Keeping Hierarchy

TI on AES

Practical Security Evaluation

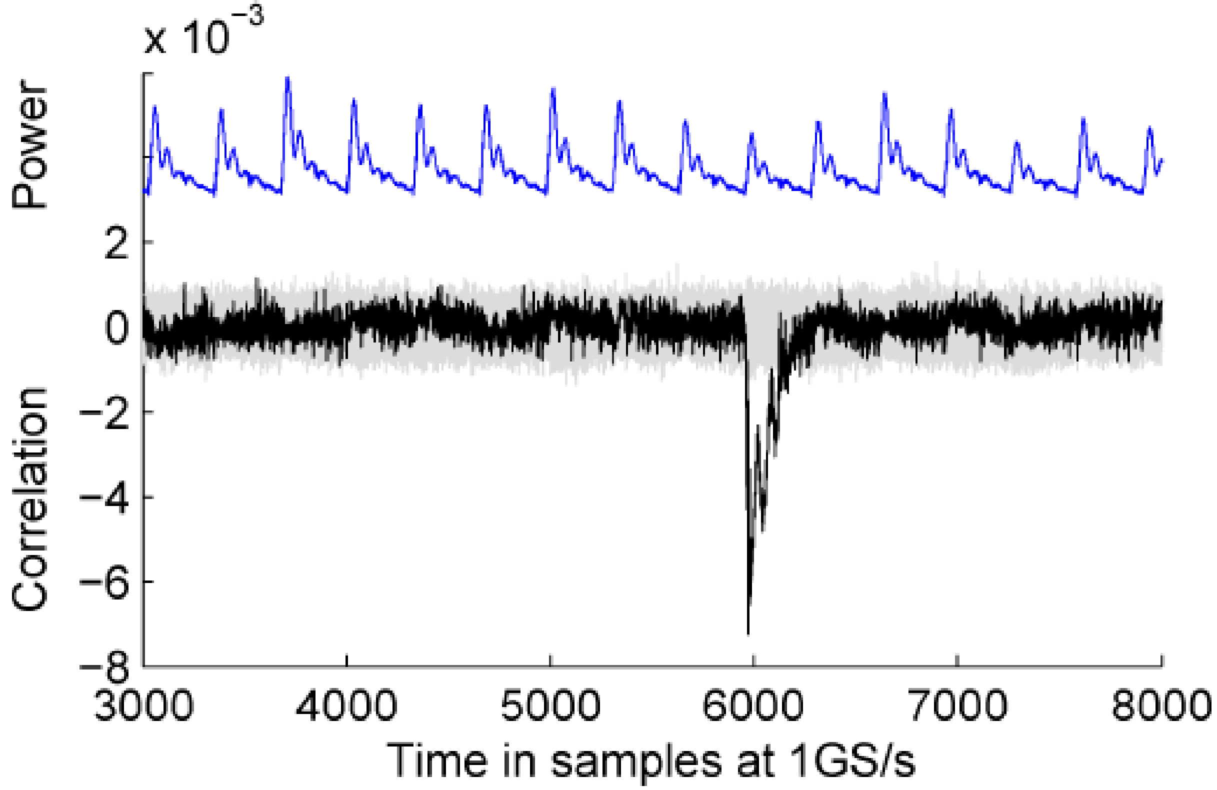
- PRNG on, first order DPA / correlation collision attack
- 10 million traces

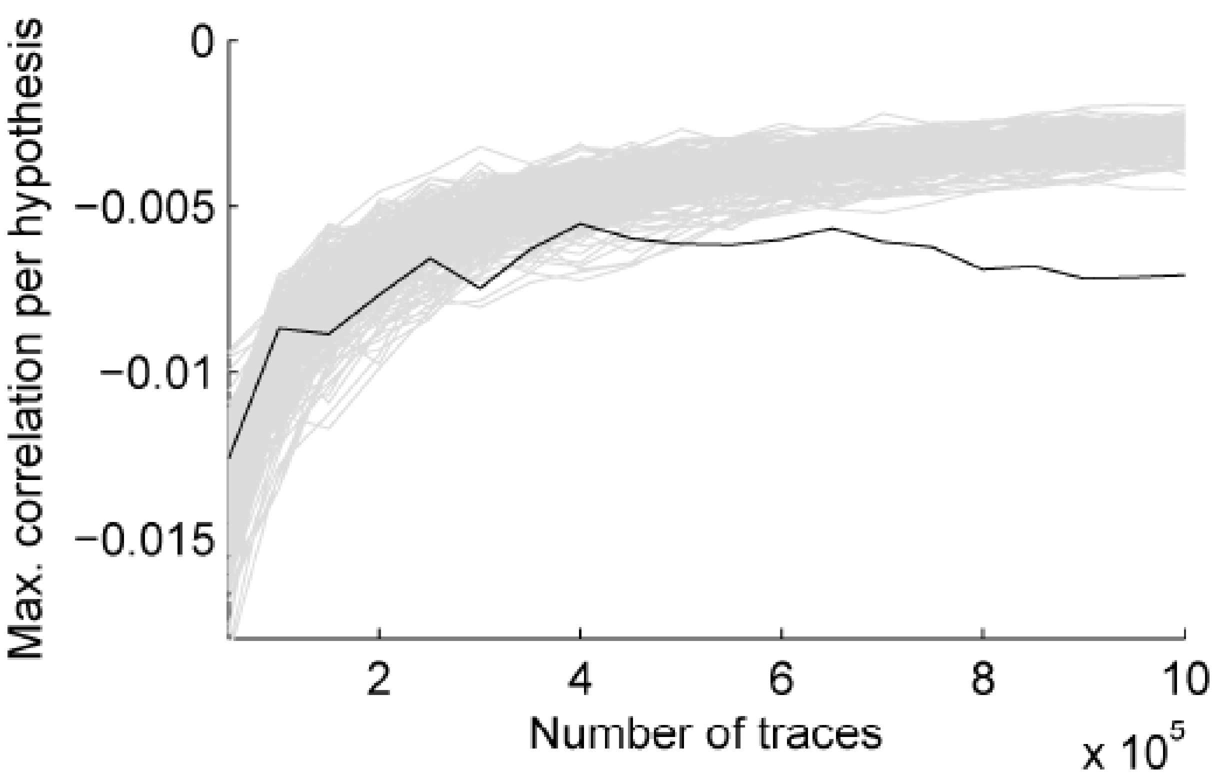


TI on AES

Practical Security Evaluation

- PRNG on, second order DPA
- HD model at S-box output



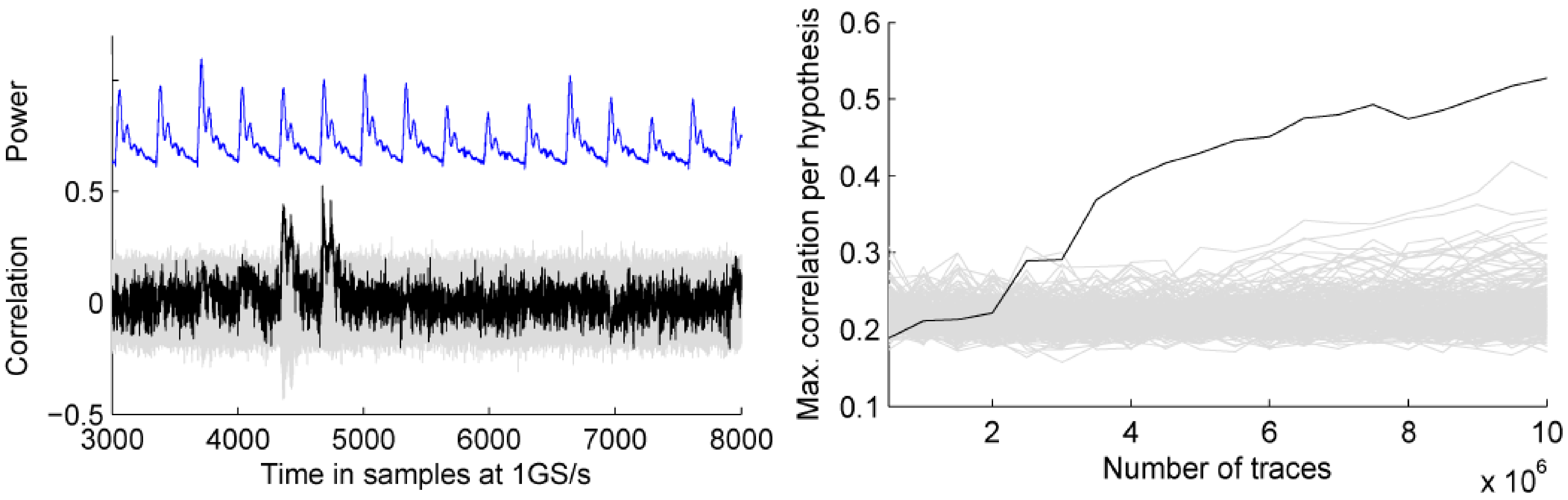


KU LEUVEN

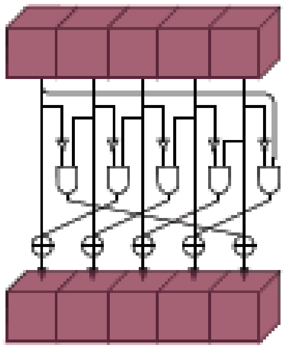
TI on AES

Practical Security Evaluation

- PRNG on, second order correlation collision attack



Applications - Keccak

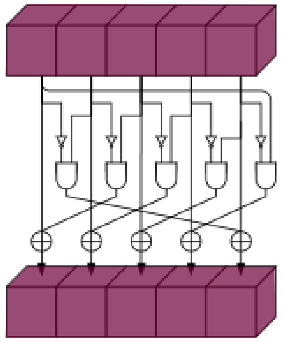


"Efficient and First-Order DPA Resistant Implementations of Keccak", B.Bilgin et al., Cardis 2013.

- uses 5x5 S-box with degree 2, thus 3 shares
- 32,6 kGE (10,6 kGE unprotected)

- Uniformity issues – how to solve?
  - Re-masking – 3200 (naive), 1280 (in χ) , 4 (in rows) bits per round
  - Find a uniform sharing (3+CT or 4 shares)
  - Ignore uniformity - the leak is too small (ongoing work)

Applications - Keccak



χ function

$x_i' \longleftarrow x_i + (x_{i+1} + 1)\, x_{i+2}$

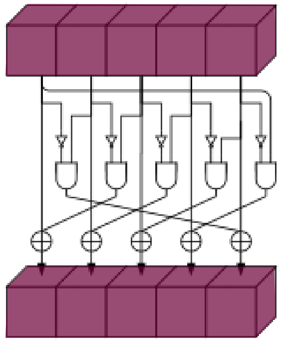
$$A_i' \leftarrow \chi_i'(B,C) \triangleq B_i + (B_{i+1}+1)B_{i+2} + B_{i+1}C_{i+2} + B_{i+2}C_{i+1},$$
$$B_i' \leftarrow \chi_i'(C,A) \triangleq C_i + (C_{i+1}+1)C_{i+2} + C_{i+1}A_{i+2} + C_{i+2}A_{i+1},$$
$$C_i' \leftarrow \chi_i'(A,B) \triangleq A_i + (A_{i+1}+1)A_{i+2} + A_{i+1}B_{i+2} + A_{i+2}B_{i+1}.$$

Not uniform

1. Inject fresh randomness to preserve uniformity
2. Find a uniform sharing

KU LEUVEN

Applications - Keccak



χ function

$x_i' \leftarrow x_i + (x_{i+1} + 1)\, x_{i+2}$

$$A_i' \leftarrow \chi_i'(B,C) \triangleq B_i + (B_{i+1}+1)B_{i+2} + B_{i+1}C_{i+2} + B_{i+2}C_{i+1},$$
$$B_i' \leftarrow \chi_i'(C,A) \triangleq C_i + (C_{i+1}+1)C_{i+2} + C_{i+1}A_{i+2} + C_{i+2}A_{i+1},$$
$$C_i' \leftarrow \chi_i'(A,B) \triangleq A_i + (A_{i+1}+1)A_{i+2} + A_{i+1}B_{i+2} + A_{i+2}B_{i+1}.$$

Not uniform

1. Inject fresh randomness to preserve uniformity
2. Find a uniform sharing

Applications - Keccak

χ function
Fresh Randomness

- Standard masking [MPLPW'11]

$$A'_i \leftarrow \chi'_i(B,C) + P_i + S_i,$$
$$B'_i \leftarrow \chi'_i(C,A) + P_i,$$
$$C'_i \leftarrow \chi'_i(A,B) + S_i,$$

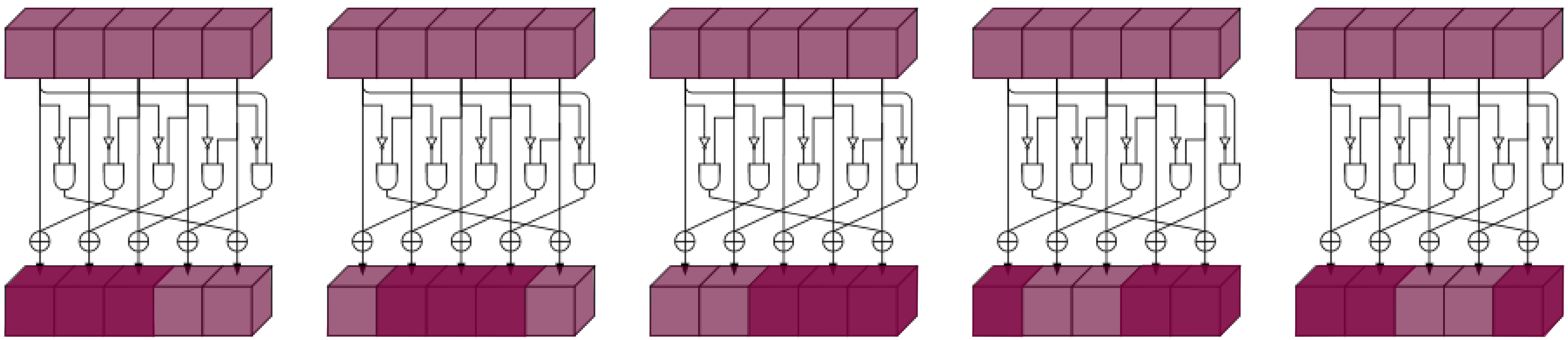
- 2 random bits per state bit
- One needs 3200 bits per round

Not feasible in practice

Applications - Keccak

χ function
Fresh Randomness

For any consecutive 3 positions, the output shares are uniform



- 4 random bits per each χ operation
- 1280 bits per round

Still too much in practice

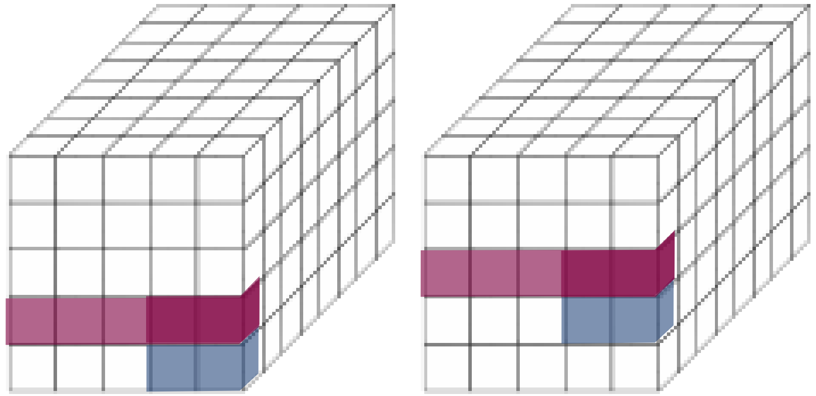
KU LEUVEN

Applications - Keccak

χ function
Fresh Randomness

Make the output row *j+1* uniform by using input from row *j*

$$A_i'^{(j)} \leftarrow \chi_i'(B^{(j)}, C^{(j)}) + A_i^{(j-1)} + B_i^{(j-1)},$$
$$B_i'^{(j)} \leftarrow \chi_i'(C^{(j)}, A^{(j)}) + A_i^{(j-1)},$$
$$C_i'^{(j)} \leftarrow \chi_i'(A^{(j)}, B^{(j)}) + B_i^{(j-1)},$$

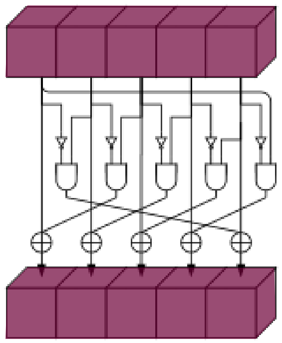


To break circular dependency, use fresh masks in one row

Detailed proof in the paper

- 4 random bits per round
- 96 bits in total for 24 rounds of KECCAK-*f*

Applications – Keccak



χ function

$x_i' \leftarrow x_i + (x_{i+1} + 1)\, x_{i+2}$

$$A_i' \leftarrow \chi_i'(B, C) \triangleq B_i + (B_{i+1} + 1)B_{i+2} + B_{i+1}C_{i+2} + B_{i+2}C_{i+1},$$
$$B_i' \leftarrow \chi_i'(C, A) \triangleq C_i + (C_{i+1} + 1)C_{i+2} + C_{i+1}A_{i+2} + C_{i+2}A_{i+1},$$
$$C_i' \leftarrow \chi_i'(A, B) \triangleq A_i + (A_{i+1} + 1)A_{i+2} + A_{i+1}B_{i+2} + A_{i+2}B_{i+1}.$$

Not uniform

1. Inject fresh randomness to preserve uniformity
2. Find a uniform sharing

Threshold Implementations

χ function

Uniform Sharing

- x With 3 shares with different sharing functions, i.e. with correction terms
- ✓ With more shares

$$
\begin{aligned}
A'_i &\leftarrow B_i + B_{i+2} + ((B_{i+1} + C_{i+1} + D_{i+1})(B_{i+2} + C_{i+2} + D_{i+2})), \\
B'_i &\leftarrow C_i + C_{i+2} + (A_{i+1}(C_{i+2} + D_{i+2}) + A_{i+2}(C_{i+1} + D_{i+1}) + A_{i+1}A_{i+2}), \\
C'_i &\leftarrow D_i + D_{i+2} + (A_{i+1}B_{i+2} + A_{i+2}B_{i+1}), \qquad\qquad i = 0, 1, 2, 4. \\
D'_i &\leftarrow A_i + A_{i+2},
\end{aligned}
$$

$$
\begin{aligned}
A'_3 &\leftarrow B_3 + B_0 + C_0 + D_0 + ((B_4 + C_4 + D_4)(B_0 + C_0 + D_0)), \\
B'_3 &\leftarrow C_3 + A_0 + (A_4(C_0 + D_0) + A_0(C_4 + D_4) + A_0A_4), \\
C'_3 &\leftarrow D_3 + (A_4B_0 + A_0B_4), \\
D'_3 &\leftarrow A_3.
\end{aligned}
$$

Applications - Fides

Design of the crypto algorithm

Secure implementation crypto algorithm

"Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware", B.Bilgin et al, CHES 2013.

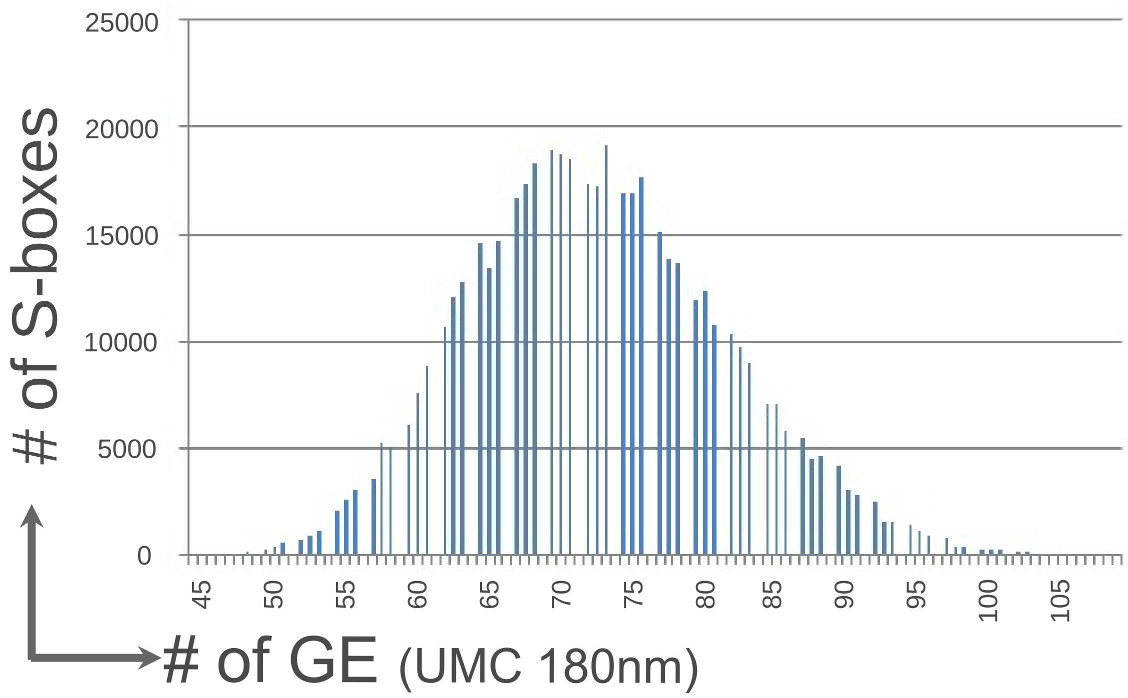
- 5x5 AB (Almost Bent);
  degree 2 (two), 3 (one), 4 (one);
- 6x6 APN (Almost Perfect Nonlinear);
  degree 4 (one); decomposition in two permutations of degree 3 and 2.
- TI with 4 shares

Applications

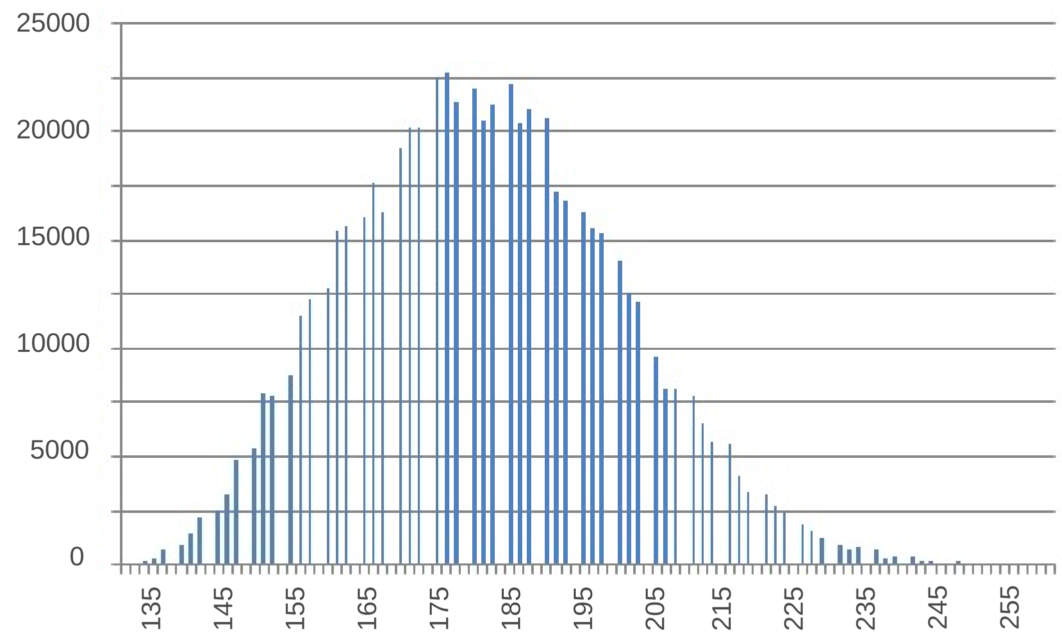
FIDES-80

Affine Equivalent to AB permutation

Unshared S-box



Shared S-box



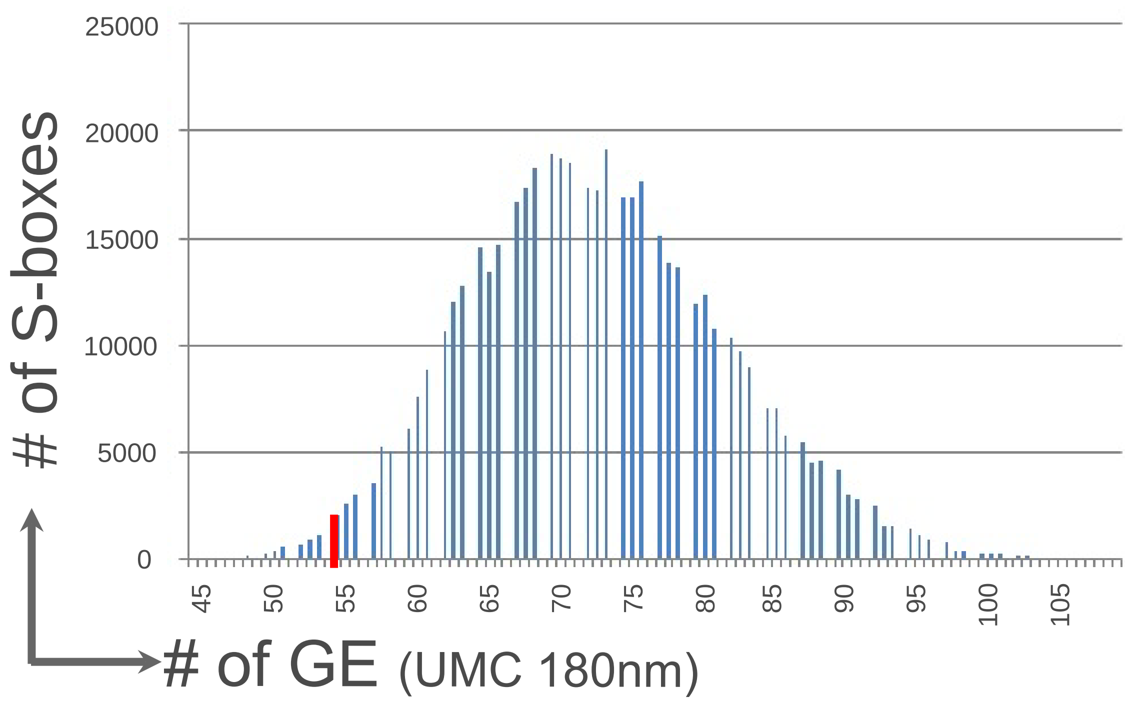
Find the best S-box

Applications

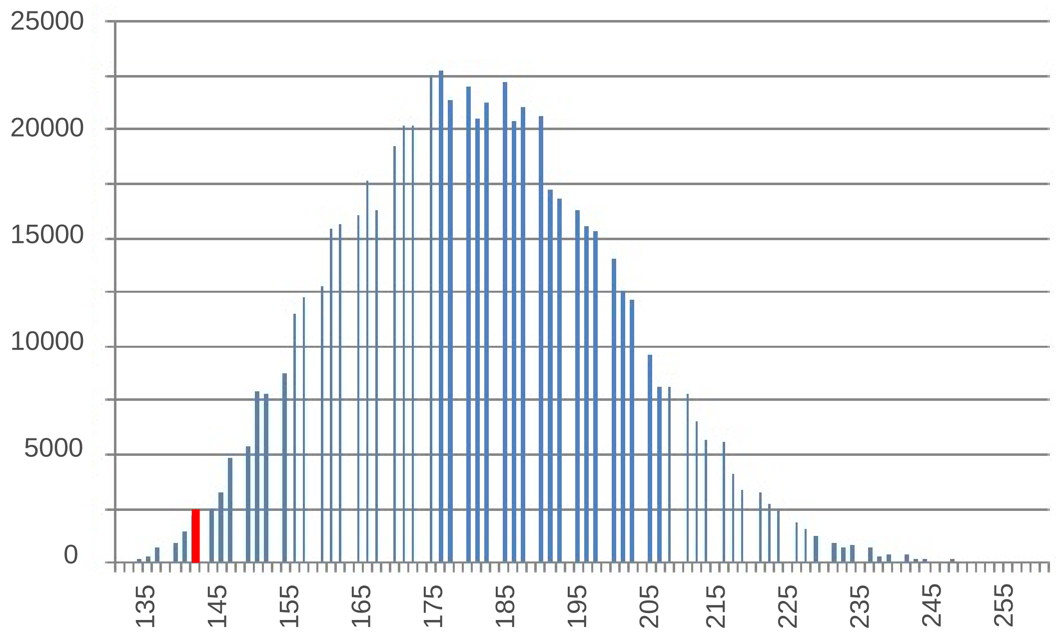
FIDES-80

Affine Equivalent to AB permutation

Unshared S-box



Shared S-box



4,2 kGE (1,1kGE unprotected)

Outline

- Threshold Implementations (update)
- Applications of TI
- Higher-order TI

Higher Order TI

(In submission, B.Bilgin et.all, 2014.)

Property 2 (d-th order non-completeness). Any combination of up to d component functions f_i of F must be independent of at least one input share.

Theorem 1. If the input masking X of the shared function F is a uniform masking and F is a d-th order TI then the d-th statistical moment of the power consumption of a circuit implementing F is independent of the unmasked input value x even if the inputs are delayed or glitches occur in the circuit.

The number of shares (input and output) increases, e.g. 2^{nd} order TI for a product s_{in}=6, s_{out}=7 or s_{in}=5, s_{out}=10;

Example: 2nd order TI

- f(x) = 1+a+bc

- 5 input shares, 10 output shares

$$
\begin{aligned}
y_1 &= 1 + a_2 + b_2c_2 + b_1c_2 + b_2c_1 & y_2 &= a_3 + b_3c_3 + b_1c_3 + b_3c_1 \\
y_3 &= a_4 + b_4c_4 + b_1c_4 + b_4c_1 & y_4 &= a_1 + b_1c_1 + b_1c_5 + b_5c_1 \\
y_5 &= b_2c_3 + b_3c_2 & y_6 &= b_2c_4 + b_4c_2 \\
y_7 &= a_5 + b_5c_5 + b_2c_5 + b_5c_2 & y_8 &= b_3c_4 + b_4c_3 \\
y_9 &= b_3c_5 + b_5c_3 & y_{10} &= b_4c_5 + b_5c_4 \,.
\end{aligned}
$$

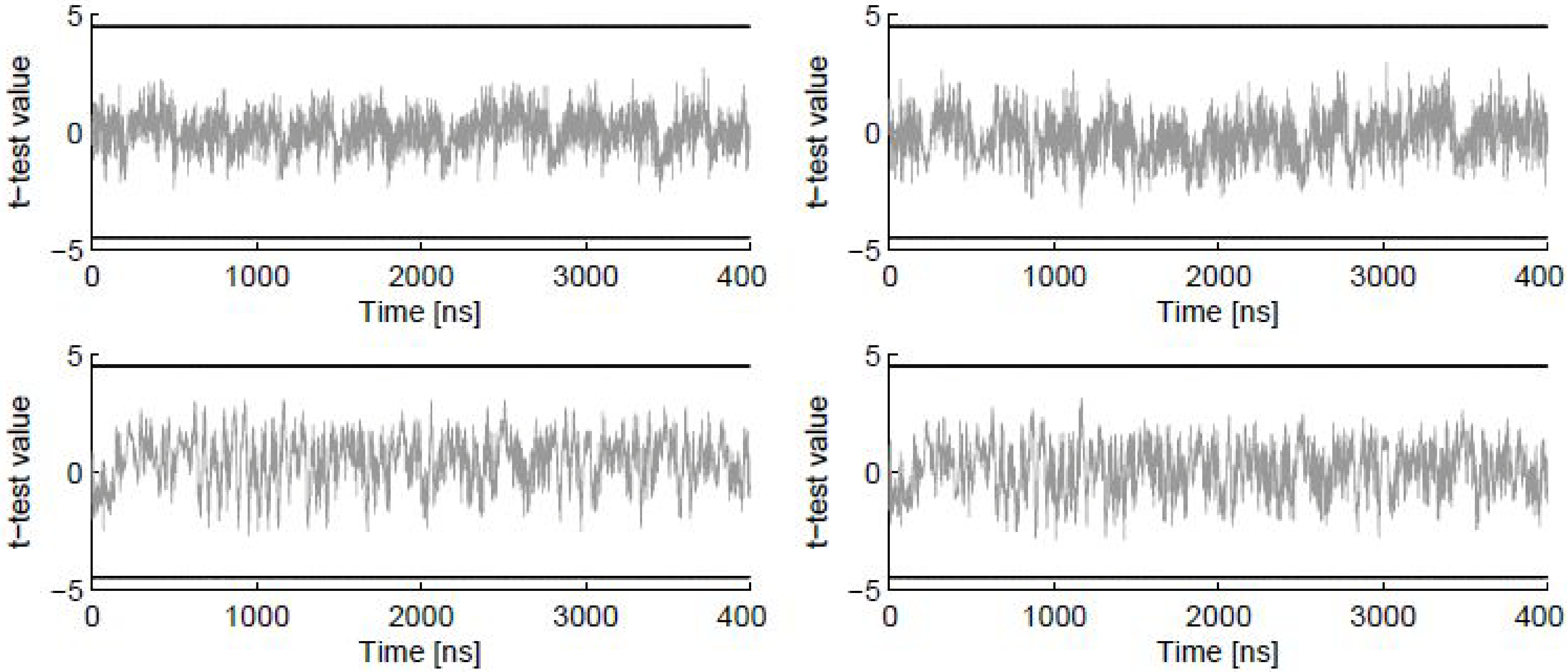
Higher Order TI – KATAN-32

- Synthesis results for plain and TI of KATAN-32

	State Array	Round Function	Key Schedule	Control	Other	Total
Plain	170	54	444	64	270	1002
1^{st} order TI	510	135	444	64	567	1720
2^{nd} order TI	900	341	444	64	807	2556
3^{rd} order TI	1330	760	444	64	941	3539

Higher Order TI – KATAN-32

- Fixed-vs-random t-test evaluation results with PRNG switched on for a randomly chosen fixed plaintext
- From top to bottom: 1st; 2nd, 3rd and 5th order statistical moment; 5 million measurements.



KU LEUVEN

Conclusions

- TI is provably secure against any order DPA
- TI can be efficient
- Room for improvement:
  - Solutions to uniformity problems
  - More efficient higher order DPA
  - Consider countermeasures during design process