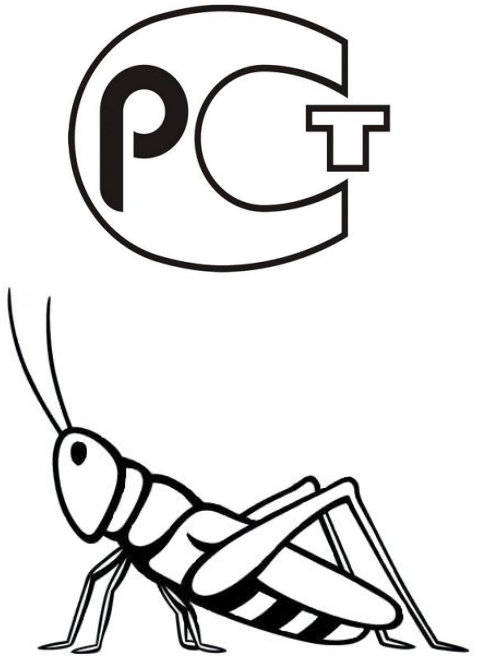
On Software Implementation of Kuznyechik on Intel CPUs

Andrey Rybkin
Andrey.Rybkin@infotecs.ru

“Kuznyechik” block cipher

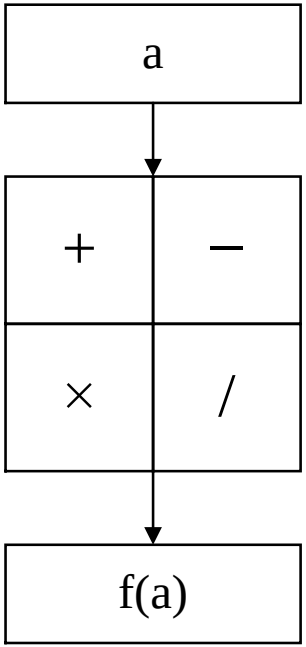
- GOST R 34.12-2015 [1]
  - block length - 128 bits
  - key length - 256 bits
- SP-network structure
  - linear transformation L
  - non-linear transformation S
  - XOR-transformation X
- Encryption algorithm
  - $E_K(a)=X[K_{10}]LSX[K_9] \dots LSX[K_2]LSX[K_1](a)$



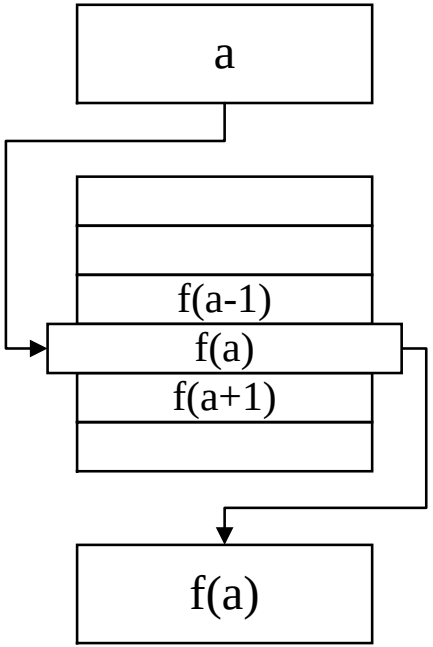
Block cipher software implementation

Lookup tables (LUTs)

arithmetic operations



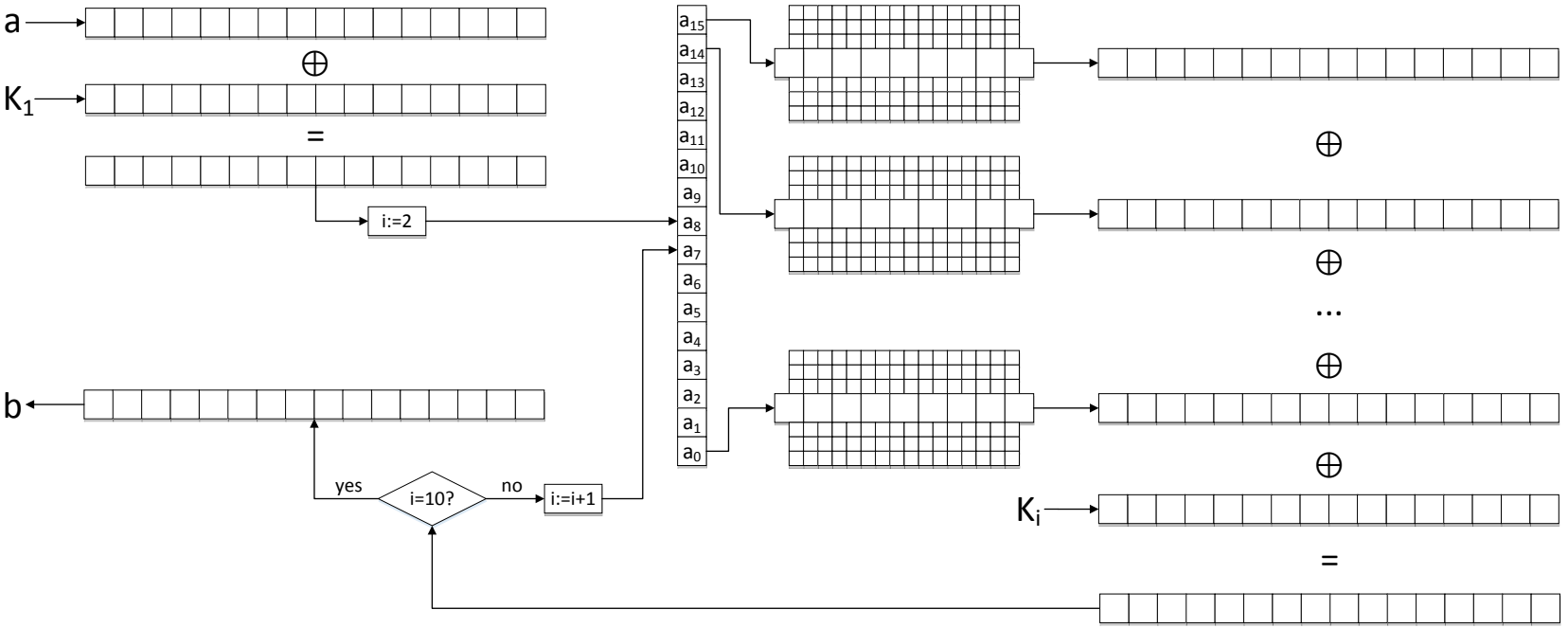
or



one lookup

Block cipher software implementation

Lookup tables (LUTs)



Main operations: lookups and XORs

Block cipher software implementation

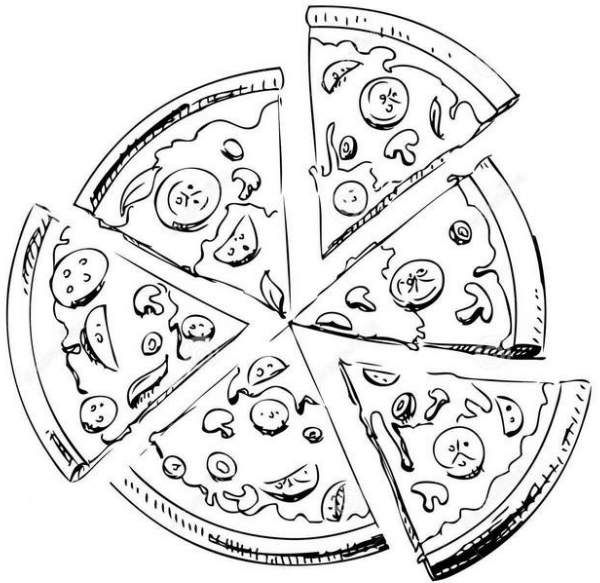
Slicing

Small data parts processing, for example

- bitwise manipulations - bit slicing
- bytewise manipulations - byte slicing

Slicing efficiency is improved in the case of

- bit- or byte-oriented cipher structure
- existence of several bits or bytes which should be processed by the same way
- ability to group these bits or bytes for parallel processing



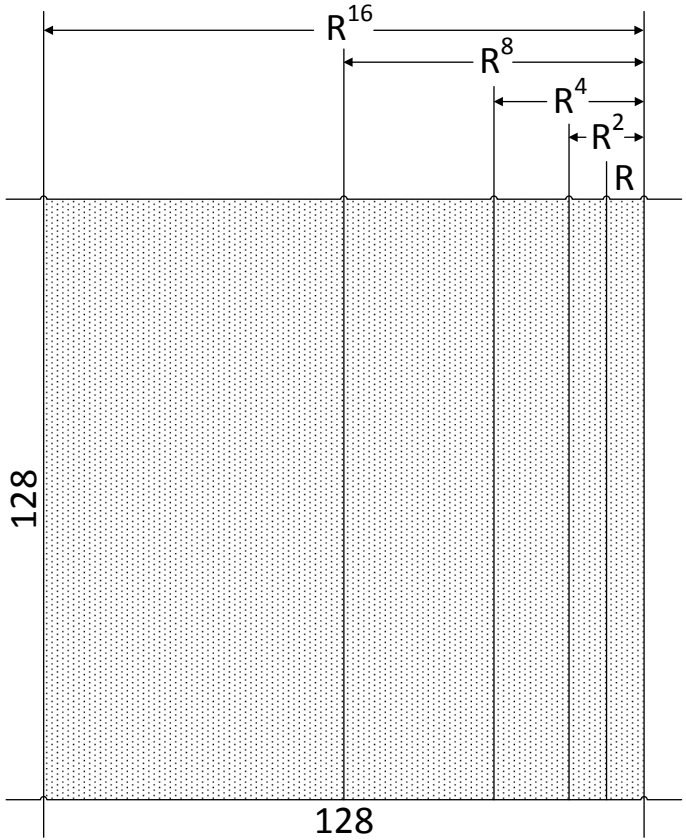
Block cipher software implementation

infotecs

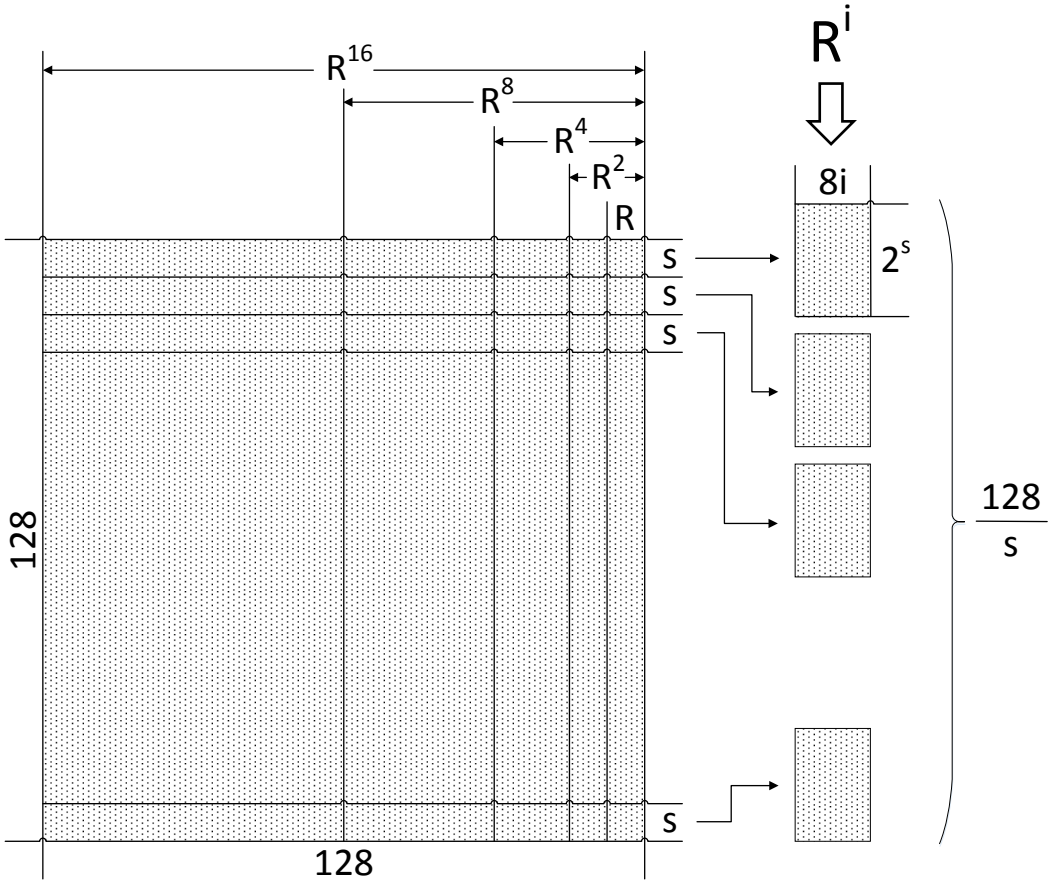
LUTs vs Slicing

	LUTs	Slicing
Possibility of combining transformations	yes ✓	no ✗
Required number of processed blocks	one or few ✓	many ✗
Data transposition	no ✓	yes ✗
Auxiliary data size	big ✗	small ✓
Constant-time execution	no ✗	yes ✓

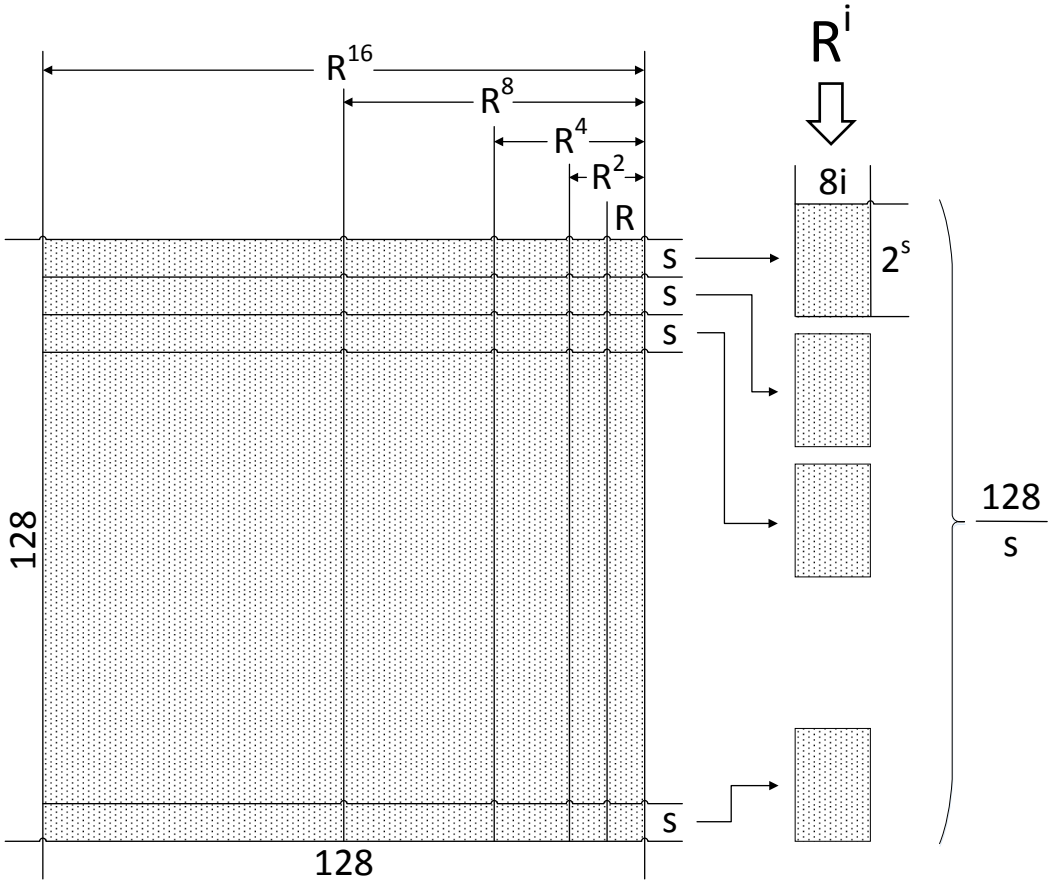
Linear transformation L



Linear transformation L



Linear transformation L



Total LUTs size: $\frac{128}{s} \cdot 2^s \cdot 8i$ bits

Number of lookups: $\frac{128}{s}$

Number of XOR: $\frac{128}{s} - 1$

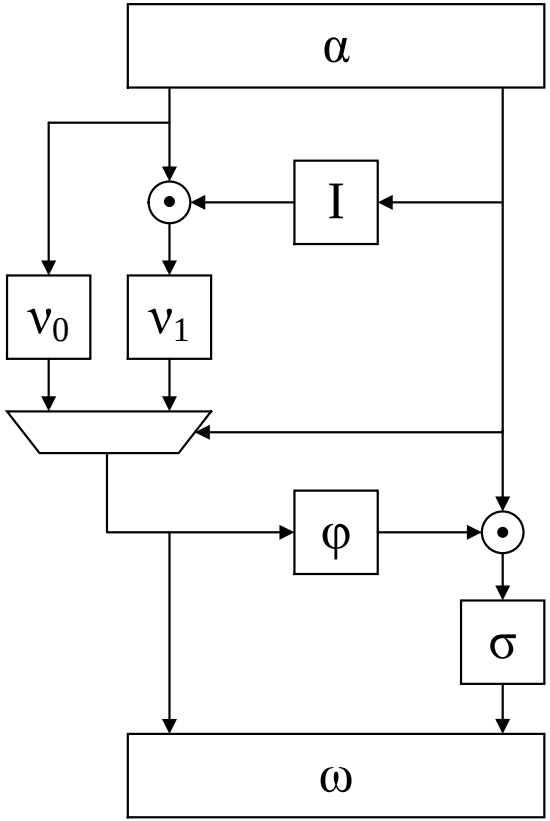
$s \uparrow \Rightarrow$ total LUTs size ↑
number of operations ↓

$s = 8$ - optimal in the general case

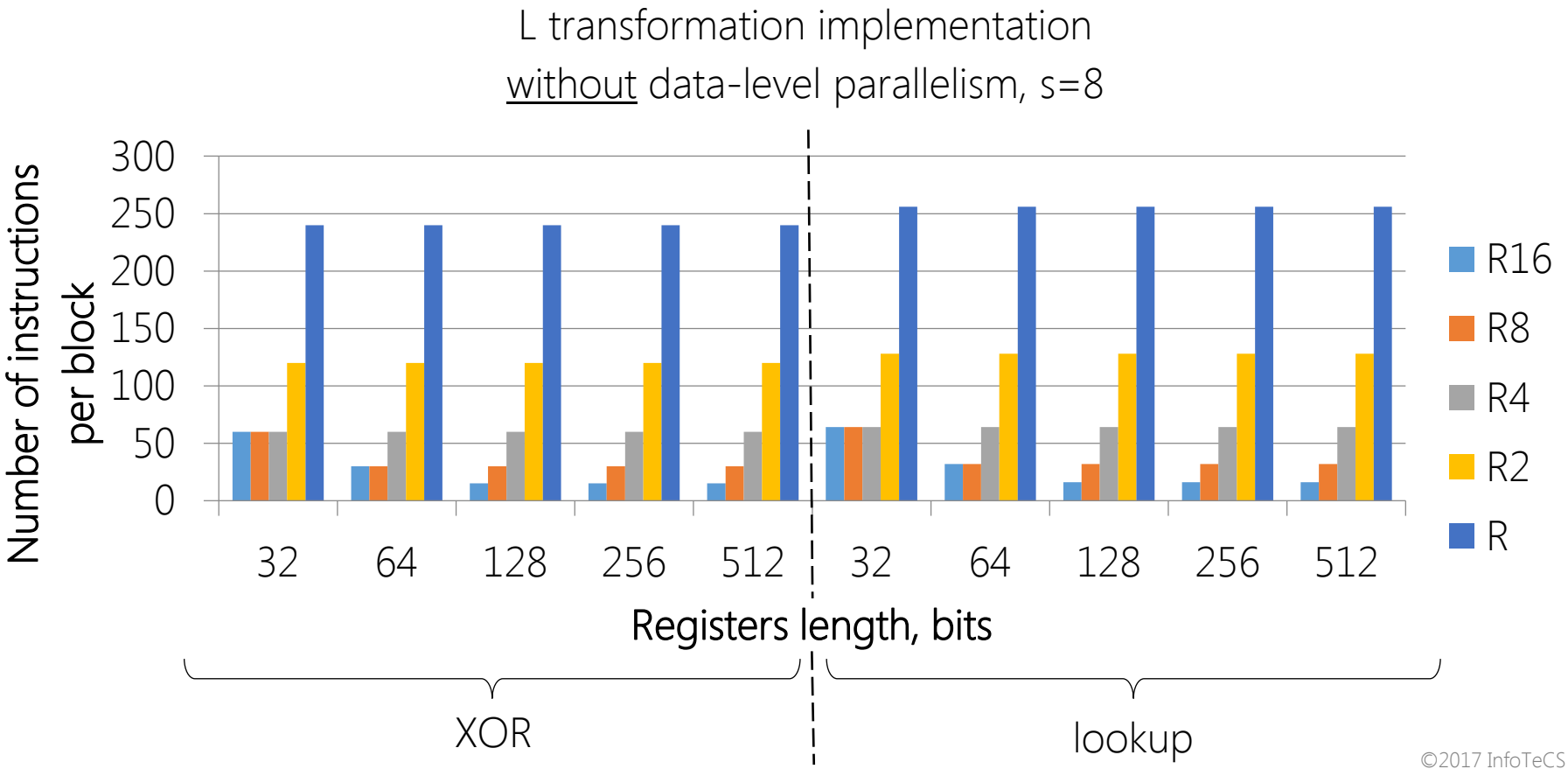
$i \uparrow \Rightarrow$ total LUTs size ↑
number of operations ?

Non-linear transformation S

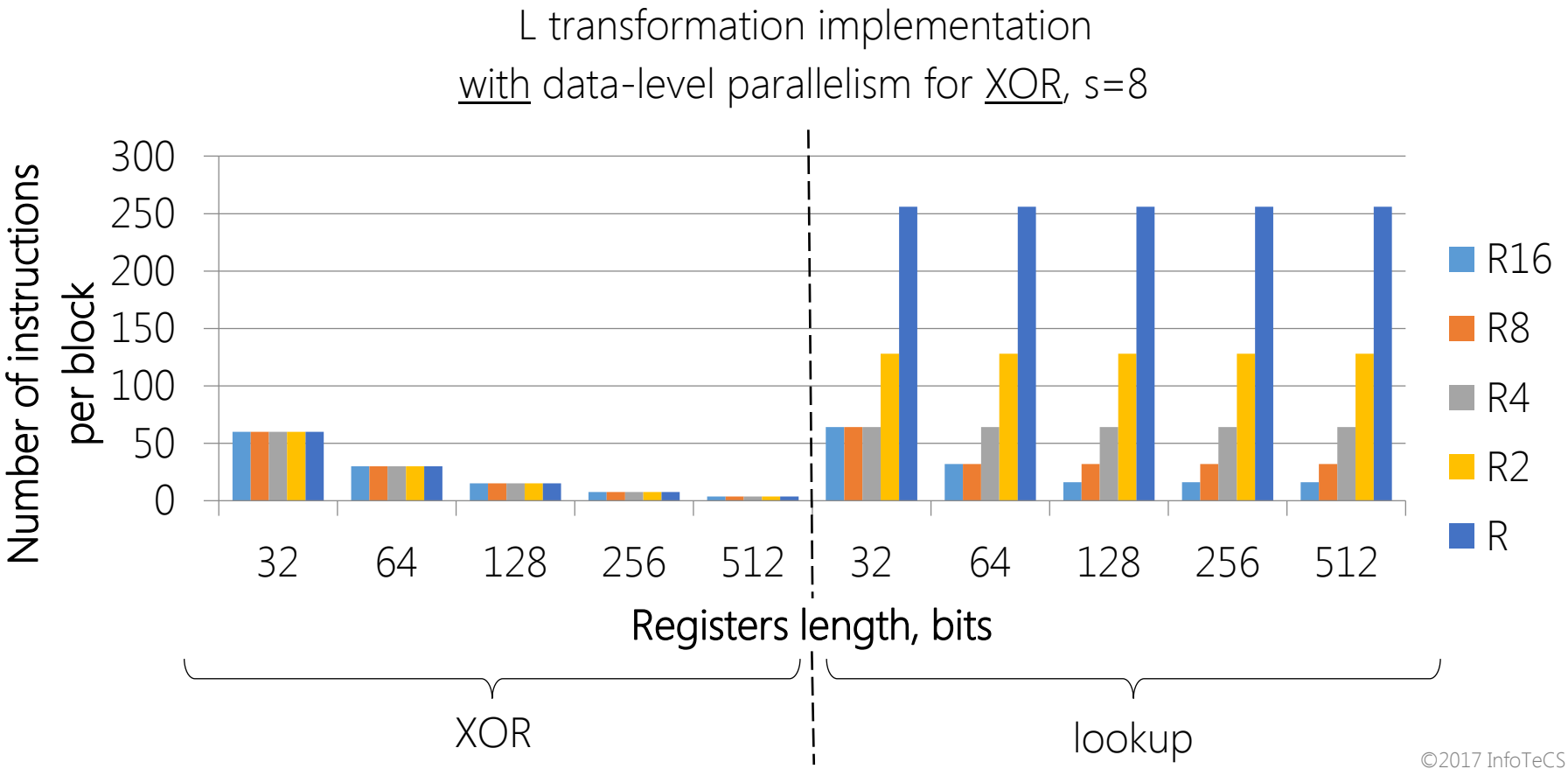
- via $V_8 \rightarrow V_8$ S-box
  - one default $V_8 \rightarrow V_8$ S-box
- via $V_4 \rightarrow V_4$ S-boxes
  - proposed in [2]
  - five $V_4 \rightarrow V_4$ non-linear transformations
  - two $V_8 \rightarrow V_8$ linear transformations
  - two $GF(2^4)$ field multiplications
- simultaneously with linear transformation
  - only if S precedes R^i
  - only in the case of using LUTs for R^i with s=8m
  - LUTs for R^i and LUTs for $R^i S$ are different



Data-level parallelism



Data-level parallelism

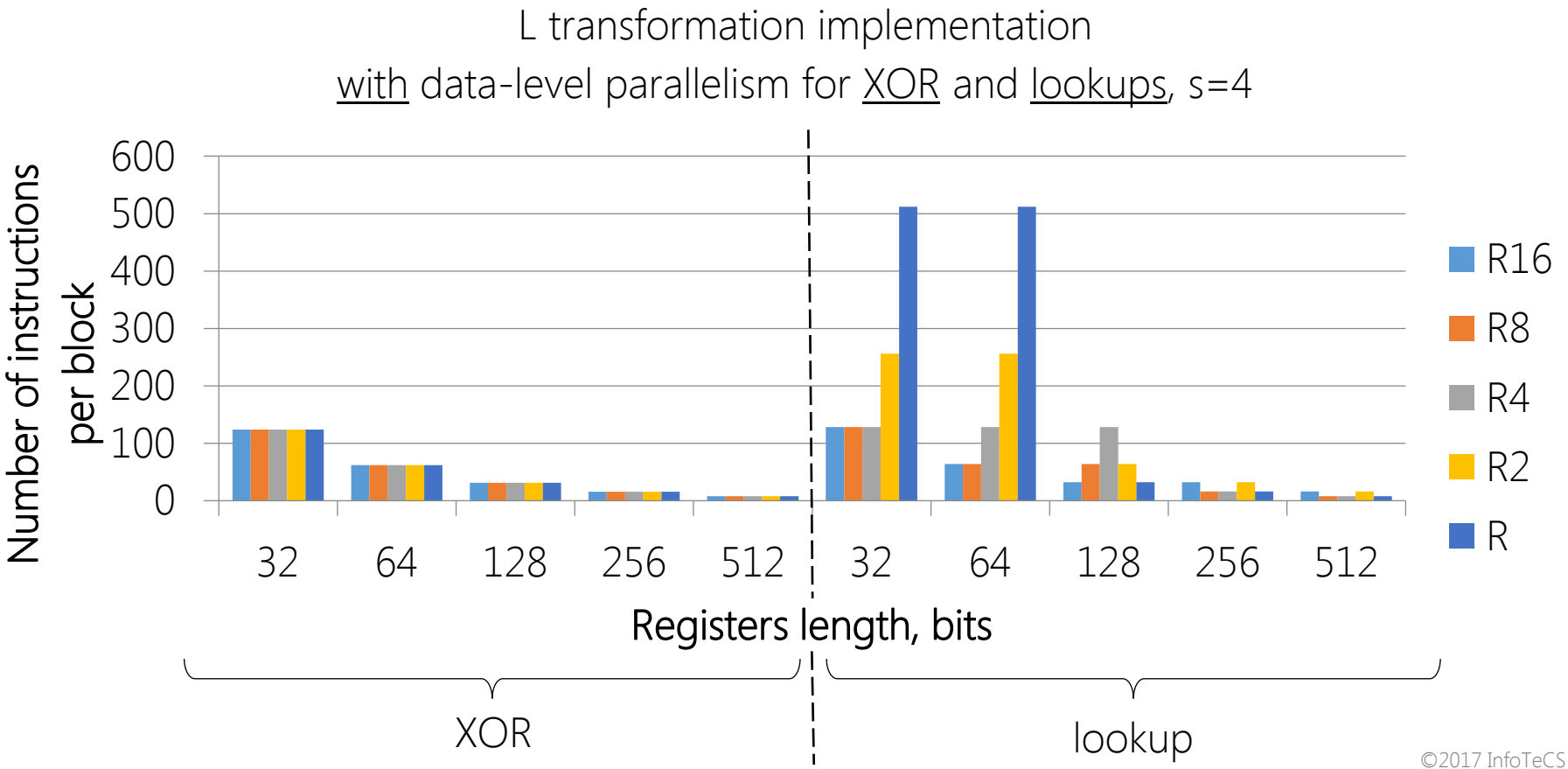


Data-level parallelism

SIMD instructions can be used to achieve data-level parallelism on Intel CPUs

Description					Implementation	
Operation	Instruction	Register bit length	Instruction set	Map	Trans-formation	s
XOR	pxor	128	SSE2			
	vpxor	256	AVX2			
Table lookup	pshufb	128	SSSE3	$V_4 \rightarrow V_8$	R	s=4
	vpshufb	256	AVX2	$V_4 \rightarrow V_8$		
	vpgatherdd	256	AVX2	$V_{32} \rightarrow V_{32}$	R^4	s=8
	vpgatherqd	256	AVX2	$V_{64} \rightarrow V_{32}$		
	vpgatherdq	256	AVX2	$V_{32} \rightarrow V_{64}$	R^8	
	vpgatherqq	256	AVX2	$V_{64} \rightarrow V_{64}$		

Data-level parallelism



Instruction-level parallelism

infotecs

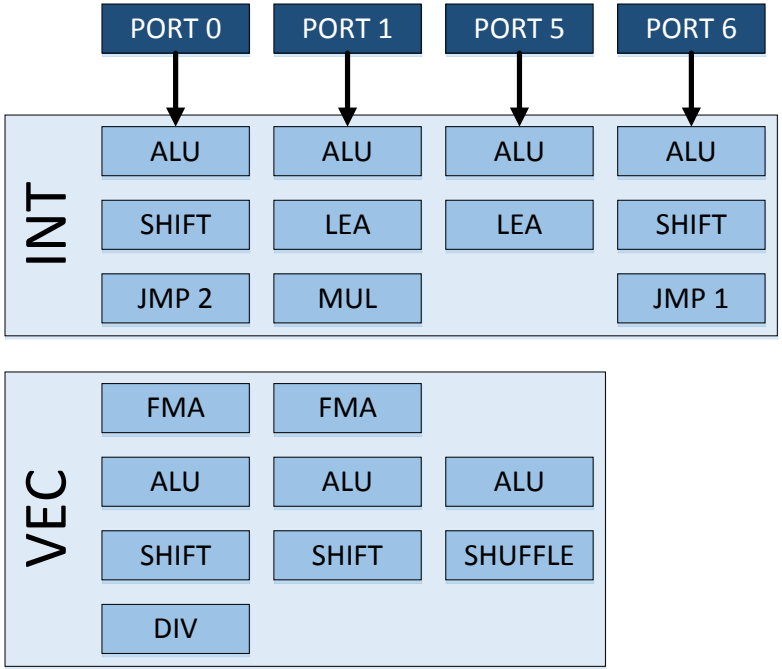
However in practice it's not all that simple

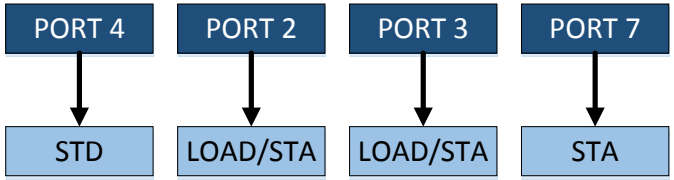
Every instruction has its own performance measures such as

- latency
- reciprocal throughput

Operation	Instruction	Latency	Reciprocal throughput
XOR	pxor, vpxor	1	0.33
Table lookup	pshufb, vpshufb	1	1
	vpgatherdd	~22	~5
	vpgatherqq	~20	~4

Instruction-level parallelism





There are several execution units into one CPU core. These units are clustered around few execution ports

There are few simple ways to increase an efficiency of EU utilization among which are:

- separation of variables corresponded to independent data parts
- instructions reordering
- independent several blocks processing

Implementations description

Name	SIMD	Transformations by LUTs	s	LUTs size (KBytes)	Parallelism degree	
					Data	Instruction
GPR-LS-1	-	LS	8	64	1	1
GPR-R8S-S-1	-	R^8S, S^{-1}	8	32+0.25	1	1
GPR-R8S-S-2	-	R^8S, S^{-1}	8	32+0.25	1	2
GPR-R4-R4S-1	-	R^4, R^4S	8	16+16	1	1
SSE-LS-1	SSE2	LS	8	64	1	1
SSE-LS-4	SSE2	LS	8	64	1	4
SSE-R8S-S-1	SSE2	R^8S, S^{-1}	8	32+0.25	1	1
AVX-LS-2	AVX2	LS	8	64	2	1
AVX-LS-8	AVX2	LS	8	64	2	4
AVX-R4-R4S-8	AVX2	R^4, R^4S	8	16+16	8	1
AVX-R-S-32	AVX2	R, S	4	0.5+0.25	32	1
AVX-R-S*-32	AVX2	R, S*	4	0.5+0.12	32	1

* in accordance to [2]

CPUs description

CPU	Generation	Number of cores (threads)	Frequency (GHz)		Cache size (KBytes)		
			base	max	L1d	L1i	L2
Intel Core i7-2600	2nd (Sandy Bridge)	4 (8)	3.40	3.80	4 x 32	4 x 32	4 x 256
Intel Core i7-4770	4th (Haswell)	4 (8)	3.40	3.90	4 x 32	4 x 32	4 x 256
Intel Core i7-6700	6th (Skylake)	4 (8)	3.40	4.00	4 x 32	4 x 32	4 x 256
Intel Xeon E5-1650 v4	5th (Broadwell)	6 (12)	3.60	4.00	6 x 32	6 x 32	6 x 256
Intel Xeon E5-2650 v4	5th (Broadwell)	12 (24)	2.20	2.90	12 x 32	12 x 32	12 x 256

Measurement method

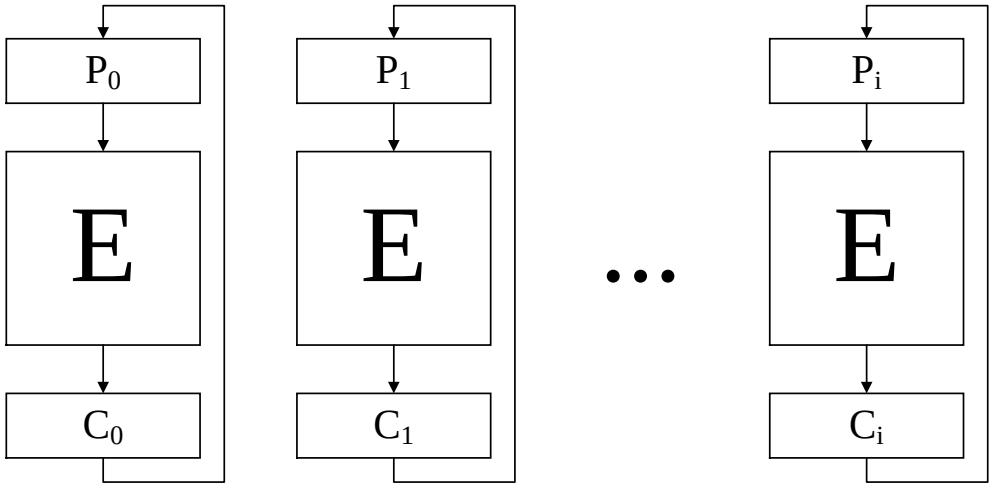
Compilers:

- Visual C++ 2015 (Windows)
- Intel C++ 17.0 (Windows)
- gcc 4.7.2 (Debian GNU/Linux)

Conditions:

- Re-keying - NO
- Key expansion - NO
- Plaintext length - up to 10^8 blocks

Encryption “mode”



Speed results in one thread on one core (MBytes/s)

CPU → Compiler ↘ Name ↓	Core i7-2600			Core i7-4770			Core i7-6700	Xeon E5-1650 v4	Xeon E5-2650 v4
	VC++	Intel	gcc	VC++	Intel	gcc	VC++	gcc	gcc
GPR-LS-1	110	125	100	135	145	125	145	120	75
GPR-R8S-S-1	130	125	120	160	160	140	170	140	85
GPR-R8S-S-2	160	155	125	210	210	155	235	150	95
GPR-R4-R4S-1	65	70	55	95	95	65	105	65	40
SSE-LS-1	145	150	140	160	165	150	170	150	90
SSE-LS-4	230	225	210	300	290	270	360	260	160
SSE-R8S-S-1	95	100	105	120	125	115	125	110	70
AVX-LS-2	-	-	-	200	170	195	210	185	110
AVX-LS-8	-	-	-	255	270	275	290	265	160
AVX-R4-R4S-8	-	-	-	65	65	45	115	65	40
AVX-R-S-32	-	-	-	235	250	220	255	205	130
AVX-R-S*-32	-	-	-	300	-	-	-	-	-

Speed results in one thread on one core

- Fastest implementations:
  - GPR-R8S-S-1, SSE-LS-1 for non-parallelizable mode - 170 MB/s
  - SSE-LS-4 for parallelizable mode - 360 MB/s
- Optimal transformations for small number of blocks:
  - General Purpose Registers - R^8S and S^{-1}
  - SSE Registers - LS
  - AVX Registers - LS
- ”vpgatherdd” instruction (AVX-R4-R4S-8 - 115 MB/s):
  - it is not effective because of its large latency and reciprocal throughput
- ”vpshufb” instruction (AVX-R-S*-32 - 300 MB/s):
  - it allows to use register length effectively (transition to 512 bit registers)
  - it allows to obtain a high-speed L implementation (about 500 MB/s without S)
  - it allows to obtain high-speed cache-timing resistant implementations

Speed results in one thread on one core

- Platforms
  - transition from 2nd to 4th generation leads to 20-25% speed-up on average
  - transition from 4th to 6th generation leads to 5-10% speed-up on average
  - speed-ups are nearly proportional to instruction-level parallelism degrees
  - ”vpgatherdd” instruction is speeded up by 75%
- Compilers
  - if instruction-level parallelism is not used then Intel C++ is slightly more efficient than Visual C++
  - if instruction-level parallelism is used then Intel C++ is slightly less efficient than Visual C++
  - gcc is less efficient than Intel C++ and Visual C++ in most cases (except AVX-LS-8)

Speed comparison in one thread on one core

infotecs

Paper	Processor	MBytes/s	Cycles/Byte	Notes
[3]	Core i7-2600 @ 3.4 GHz	138	23.5 - 26.3	non-parallelizable mode
[4]	Core i5-2500K @ 3.3 GHz	135	23.3 - 26.1	ECB mode
		129	24.4 - 27.4	CTR mode
[5]	Core i5-6500 @3.2 GHz	335	9.1 - 10.2	CTR mode
This paper	Core i7-6700 @ 3.4 GHz	360	9 - 10.6	parallelizable mode
		170	19.1 - 22.4	non-parallelizable mode
	Core i7-4770 @ 3.4 GHz	300	10.8 - 12.4	byte slicing implementation

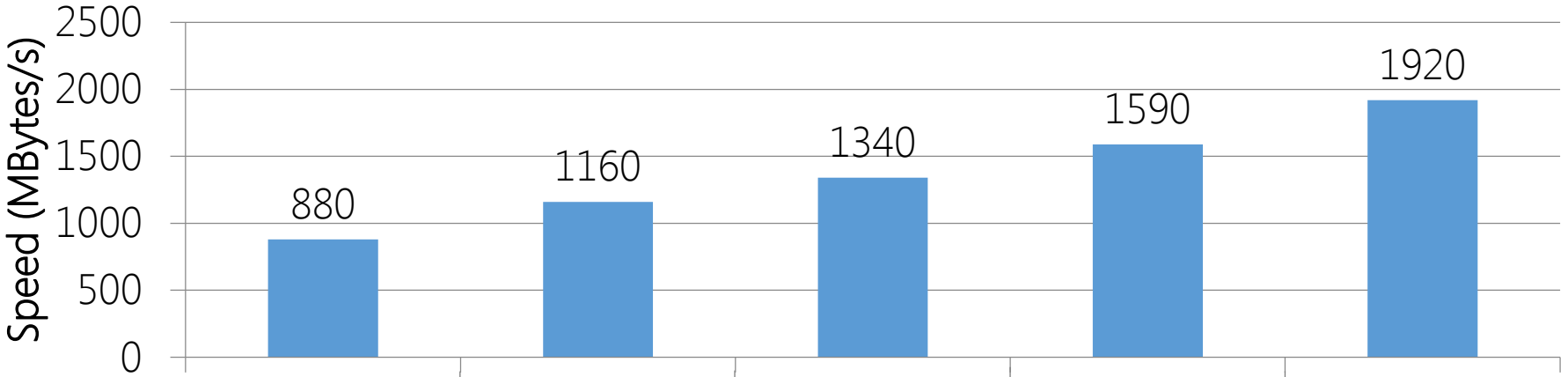
Speed results on several cores (MBytes/s)

CPU → Cores ↘ Name ↓	Core i7-2600			Core i7-4770			Core i7-6700		Xeon E5-1650 v4			Xeon E5-2650 v4		
	1	P	L	1	P	L	1	P	1	P	L	1	P	L
GPR-R8S-S-1	130	120	60	160	155	75	-	-	140	140	70	85	85	40
GPR-R8S-S-2	160	150	60	210	200	75	-	-	150	150	60	95	95	35
SSE-LS-1	145	135	80	160	155	85	-	-	150	150	85	90	90	50
SSE-LS-4	230	220	90	300	290	105	360	335	260	260	100	160	160	60
AVX-LS-2	-	-	-	200	195	100	-	-	185	185	90	110	110	55
AVX-LS-8	-	-	-	255	240	100	-	-	265	265	100	160	160	60

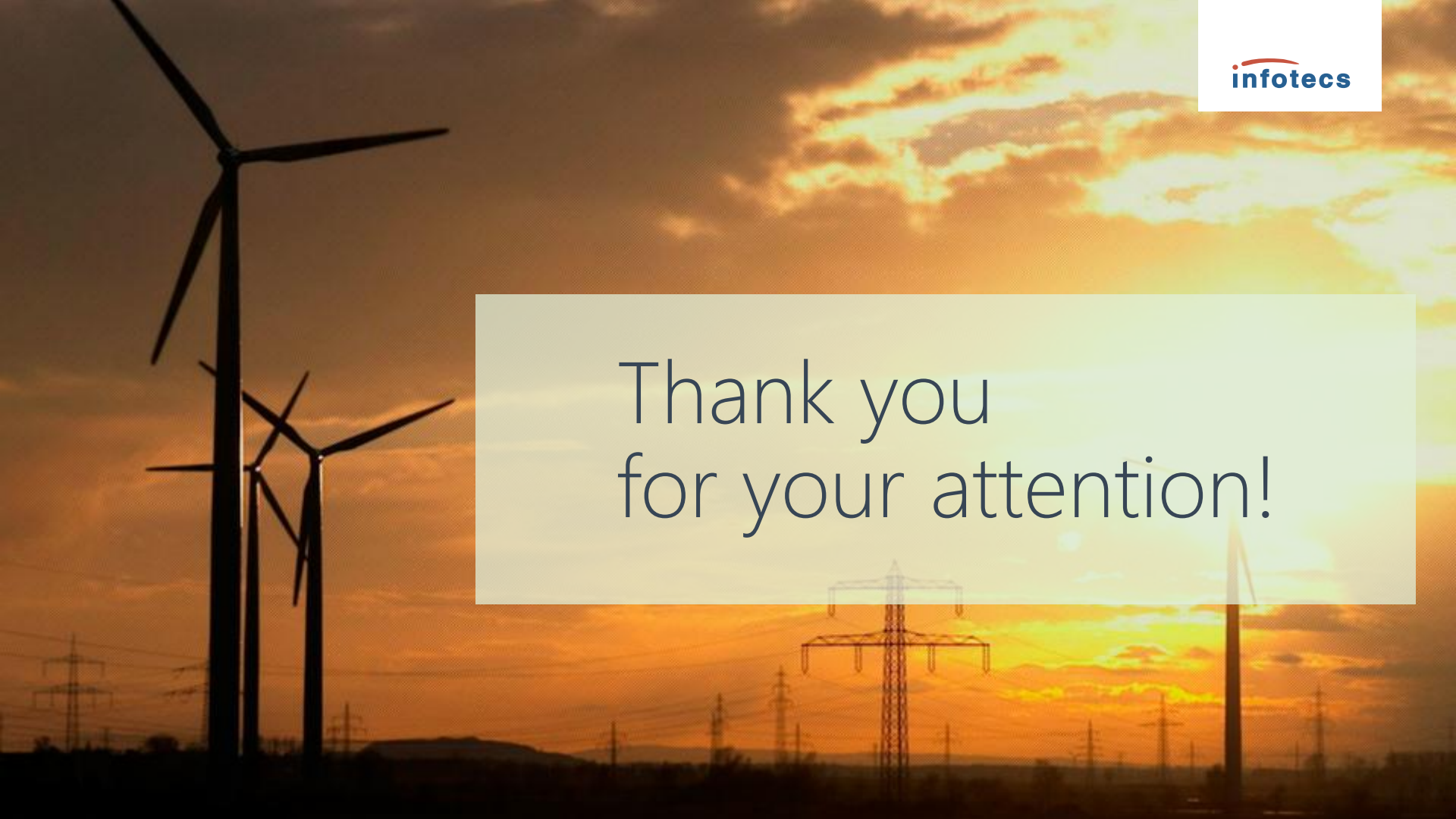
Speed results on several cores

- Turbo Boost impact
  - On Intel Core processors there is 3-8% slow-down
    if all physical cores are utilized
  - On Intel Xeon processors there is not slow-down
    if all physical cores are utilized
- Reasonability of using hyper-threading
  - if instruction-level parallelism is not used
    then hyper-threading lead to about 2 times slow-down (except SSE-LS-1)
  - if instruction-level parallelism is used
    then hyper-threading lead to 2.5-3 times slow-down

Cents / megabytes per second



Core i7-2600	Core i7-4770	Core i7-6700	Xeon E5-1650 v4	Xeon E5-2650 v4
29.7 $\frac{cents}{MB/s}$	26.6 $\frac{cents}{MB/s}$	23.0 $\frac{cents}{MB/s}$	38.9 $\frac{cents}{MB/s}$	60.8 $\frac{cents}{MB/s}$


infotecs
Thank you
for your attention!

References

[1] GOST R 34.12-2015, “National standard of the Russian Federation. Information technology. Cryptographic data security. Block ciphers", 2015.

[2] Biryukov A., Perrin L., Udovenko A., “Reverse-Engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1", Lecture Notes in Computer Science, EUROCRYPT 2016, 9665, Springer, Berlin, Heidelberg, 2016, 372-402.

[3] Borodin M. A., Rybkin A. S., “High-speed software implementation of the Kuznyechik block cipher”, Information Security Problems. Computer Systems, 3 (2014), 67-73.

References

[4] Alekseev E. K., Popov V. O., Prokhorov A. S., Smyshlyaev S. V., Sonina L. A., “On the performance of one perspective LSX-based block cipher”, Матем. вопр. криптогр., 6:2 (2015), 7–17.

[5] Ahmetzyanova L., Alekseev E., Oshkin I., Smyshlyaev S., Sonina L., “On the properties of the CTR encryption mode of the Magma and Kuznyechik block ciphers with re-keying method based on CryptoPro Key Meshing”, Pre-proceedings of 5th Workshop on Current Trends in Cryptology, CTCrypt 2016 (June 6-8, 2016, Yaroslavl, Russia), 42-54.