Advanced Logic Synthesis SAT Attack Report

Jie-Hong Liu | 劉杰閎
jiehong0914@gmail.com
College of Semiconductor Research, National Tsing Hua University
Hsinchu, Taiwan

0. Abstract

This is the SAT-attack report on Advanced Logic Synthesis. And the author is Jie-Hong Liu from NTHU CoSR. This report includes the introduction and author's experiment on SAT-Attack tool. Nowadays, there are more and more papers are discussing about the security of ICs. Hence, there is why logic encryption exist. Logic encryption would modify an IC design such that it operates correctly only when a set of newly introduced inputs, called key input, are set to the correct values. In this report, we use the Parmod. et al.[1] proposed SAT Attack to attack and get the key of the design in academic benchmarks.

1. Introduction

Logic locking is a technique employed to protect the intellectual property of digital circuits by ensuring that the circuit design can only be utilized with an authorized key. Figure 1. shows an example of logic locking, where XOR gates and XNOR gates are used to obscure the signal. The output (y1, y2) will only be correct if the values of k1 and k2 are set to 00. By using this technique, the circuit design can be properly protected, making it difficult for malicious individuals to steal the design. The circuit will only work correctly with the correct keys, and different logic gate locking can be used as long as the user has the correct keys.

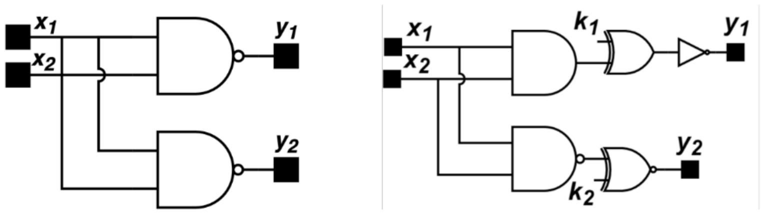


Fig. 1 Example of logic locking

2. Methodology

In this chapter, we are going to introduce the Attack Model and SAT Attack. SAT Attack is a state of art algorithm to decrypt locked circuit based on different algorithm. After using this tool, we can get the key to each benchmark.

2-1. Attack Model

In this report, we assume that the attacker can access the netlists of encrypted circuit and assume that we have the correctly activated circuits (black box) AKA oracle, we can get the model by buying the same design in the market.

2-2. SAT Attack

In the SAT Attack, attacker aims to acquire the correct key from the encrypted circuit, SAT attack would eliminate the incorrect key to the correct key by multiple iterations. After following step, the problem can be model as a SAT problem, by providing this problem to a SAT-solver, we can get the correct key without knowing "how to solve SAT".

1. Search for a distinguish input pattern (DIP) that can cause 2 different keys to generate different outputs.
2. Query oracle to obtain the correct output so that we can eliminate incorrect keys.

2-3. SAT Solver

A SAT solver is a software tool used to solve Boolean satisfiability problems. Although SAT is an NP-complete problem, which was proven by Steven and Cook [3], there are numerous fast algorithms to solve this problem.

3. Experiments

In this paragraph, we will describe our experiments applying the SAT attack on specific locked benchmarks, including 'rnd' (Random, XOR) , 'dac12' (Interference analysis, XOR), and 'sarlock/dac12' (Point Function, XOR). Each result will be recorded in the table below. The naming convention for these benchmarks is "_enc.bench". The percentage is a two-digit number that can be 05, 10, 25, or 50, and it indicates the percentage area overhead of encryption for the benchmark. Therefore, as the percentage increases, the difficulty of decrypting the circuit also increases. And the whole experiments run on CAD workstation ic21. (Memory:251.2GB)

3-1. Experiments Setup

Our experimental flow chart is shown as Figure 2. In our experiment, we use a Python script to implement total flow. We use several python modules such as 'os', 'subprocess 'to help us evaluating this tool. The os module provides a portable way of using operating system dependent functionality, while the sub-process allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. In our experiment code, there are some skills are used to get timeout exception and stop attacking the same design to attack next benchmark, i.e. in our experiment, we gave each benchmark with 2 hours to attack. The source code could be reference at the Author's GitHub repo [2].

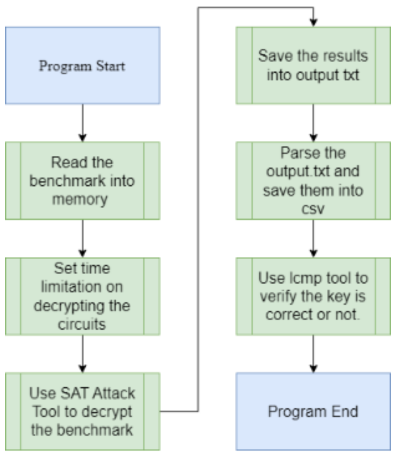

Program Start
Read the benchmark into memory
Set time limitation on decrypting the circuits
Use SAT Attack Tool to decrypt the benchmark
Save the results into output txt
Parse the output.txt and save them into csv
Use lcmp tool to verify the key is correct or not.
Program End

Fig2. Experiment Flow Chart

3-2. Experimental Result

In this paragraph, we are going to apply SAT attack on different benchmark. Table 1 is the results of applying SAT attack on DAC12/apex2, Table 2 is for RND/ex5, and Table 3 is for where the first col is the file name of each benchmark. And the number after "enc" denotes the percentage area overhead of encryption of this benchmark. "PIs" and "Pos" means the number of primary inputs and primary outputs. "Key inputs" means the number of keys, each key-inputs also means one gate inserted to performing the logic locking, Gates means there are how many gates in this benchmark. "CPU time" means this attack elapsed how much time. "Verification" use 'lcmp' tool which is also provided in SAT attack repository to verify the key is correct or not. By compare the functionality with benchmark's oracle. By these results, we can find that the PIs and POs of same filename, but different percentage benchmark is the same, while the key inputs and gates of them are different, as the percentage increase, the number of key inputs and gates increases. Since SAT attack model the attack problem as Boolean expression, then SAT solver needs iterations to remove the wrong key. By this reason, the number of key inputs would directly affect the number of SAT iterations.

file	PIs	key inputs	POs	Gates	SAT iterations	CPU time	VERIFICATION
dac12/apex2_enc05.bench	39	31	3	644	20	0.203952	TRUE
dac12/apex2_enc10.bench	39	61	3	674	25	0.36766	TRUE
dac12/apex2_enc25.bench	39	153	3	769	122	6.28487	TRUE
dac12/apex2_enc50.bench	39	305	3	925	208	29.3834	TRUE

Table 1. DAC12/apex2 results

file	PIs	key inputs	POs	Gates	SAT iterations	CPU time	VERIFICATION
rnd/ex5_enc05.bench	8	53	63	1109	14	0.157891	TRUE
rnd/ex5_enc10.bench	8	106	63	1161	35	0.444738	TRUE
rnd/ex5_enc25.bench	8	264	63	1319	42	1.096	TRUE
rnd/ex5_enc50.bench	8	528	63	1585	70	5.62259	TRUE

Table 2. RND/ex5 results

file	PIs	key inputs	POs	Gates	SAT iterations	CPU time	VERIFICATION
Sarlock/dac12/apex4_enc05.bench	10	278	19	5675	1023	228.207	TRUE
Sarlock/dac12/apex4_enc10.bench	10	546	19	5943	1023	251.73	TRUE
Sarlock/dac12/apex4_enc25.bench	10	1350	19	6747	1023	984.593	TRUE
Sarlock/dac12/dalu_enc05.bench	75	190	16	2605	TLE	TLE	-
Sarlock/dac12/dalu_enc10.bench	75	305	16	2720	TLE	TLE	-
Sarlock/dac12/dalu_enc25.bench	75	650	16	3067	TLE	TLE	-

Table 3. sarlock/dac12/ results

3-3 Evaluation and Analysis

In evaluating the results of each benchmark, we can roughly divide them into three parts: "DAC12/", "RND/", and "sarlock/DAC12/". Figure 3 represents the chart from Table 1, Figure 4 corresponds to the chart from Table 2, and Figure 5 depicts the chart from Table 3. These results indicate that the number of key inputs, SAT iterations, and CPU time exhibit linear growth.

It is important to note that in Table 3, we applied a SAT attack to the "sarlock" logic locking [4]. The results demonstrate that even though "apex4" has a significantly larger number of key inputs and gates compared to "dalu," it still managed to complete the attack within the given time limitation. The reason behind this lies in the fact that the number of primary inputs (PIs) in "dalu" is much larger than in "apex4." In the case of "sarlock" logic locking, the time required to unlock the logic locking increases exponentially. This explains why "apex4" successfully completed the attack while "dalu" did not.

Figure.3 DAC12/apex2 benchmark results

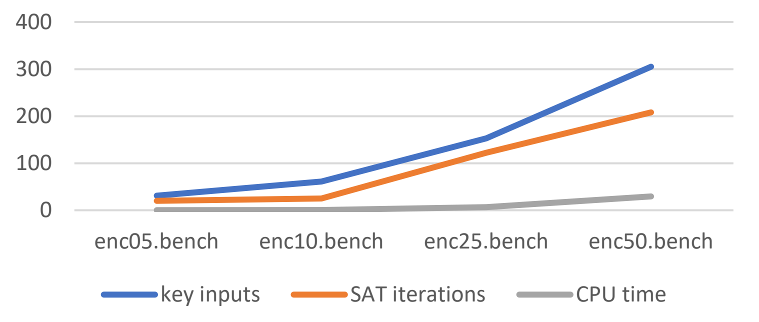


Figure 4. rnd/ex5 benchmark results

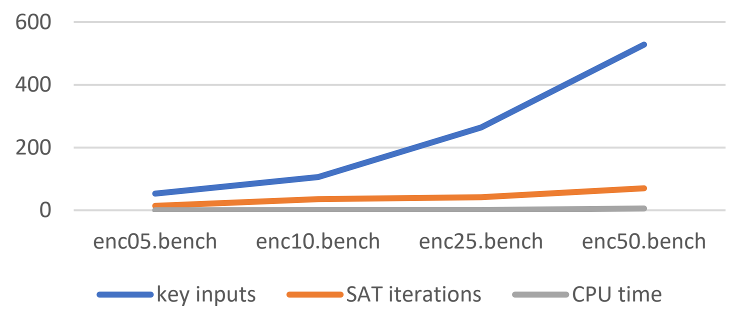


Figure.5 sarlock/apex4 benchmark results

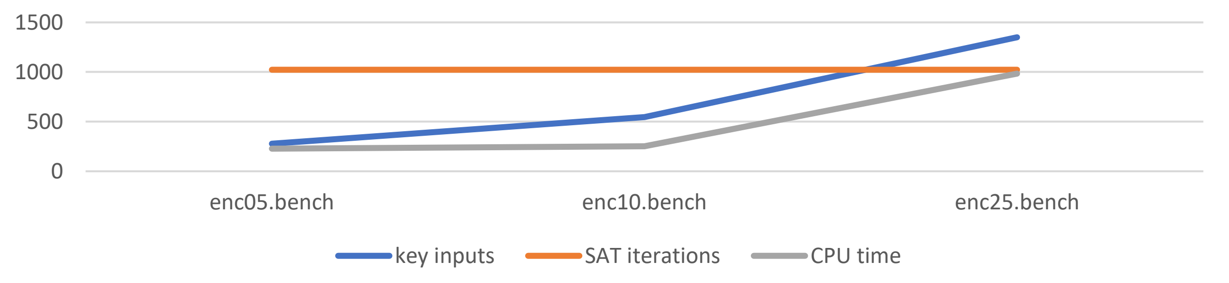
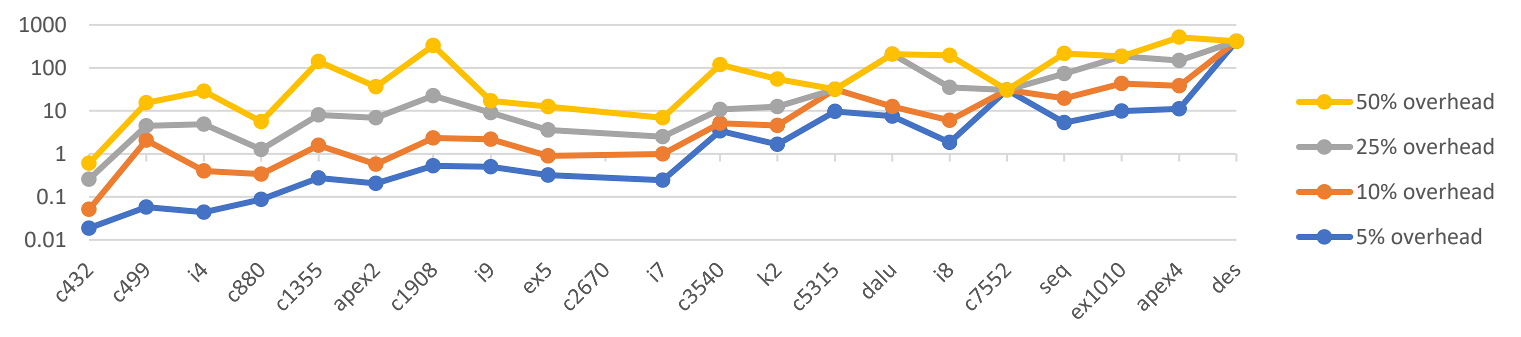


Figure. 6 CPU time v.s. benchmark (sorted benchmark) on DAC12



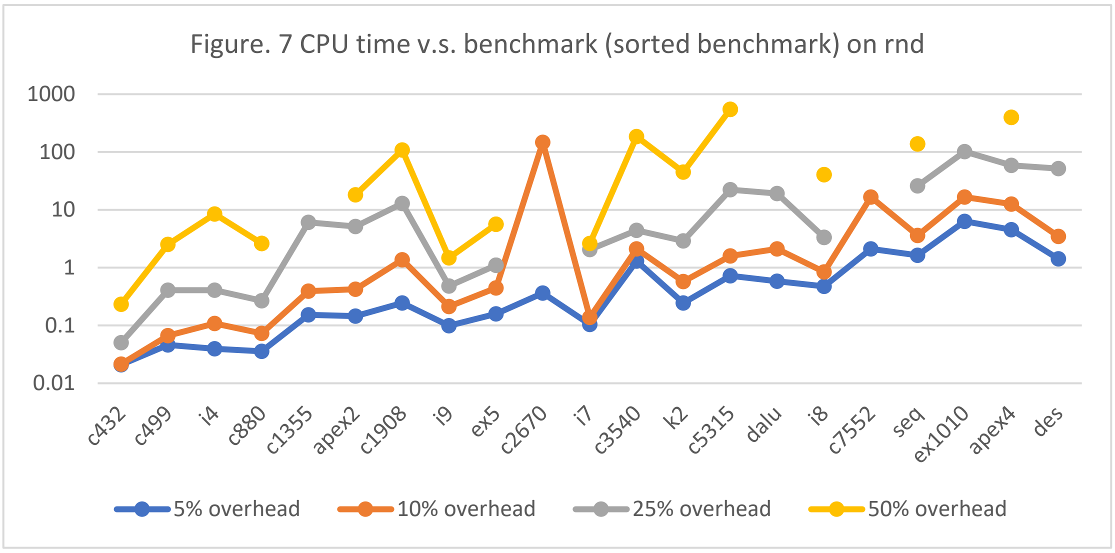


Figure. 6 and Figure. 7 shows that as the percentage overhead increase, the CPU time also increase. Some cases such as c2670 in the benchmark would be TLE since the key is too long to solve. Remember that the benchmark circuit are shown on the x-axis are sorted by gate count. The leftmost circuit is the smallest and the rightmost is the largest. The reason is because we expect that the larger circuit will take more time to decrypt, and we can see from the chart, this is almost true.

Benchmarks	Total circuits	Unlock Circuit	Ratio (#unlock/#total)	Verification rate
DAC12	84	71	84.52%	100%
RND	84	76	90.47%	100%
SARLock	33	6	18.18%	100%

Table 4. SAT attack performance on each benchmark

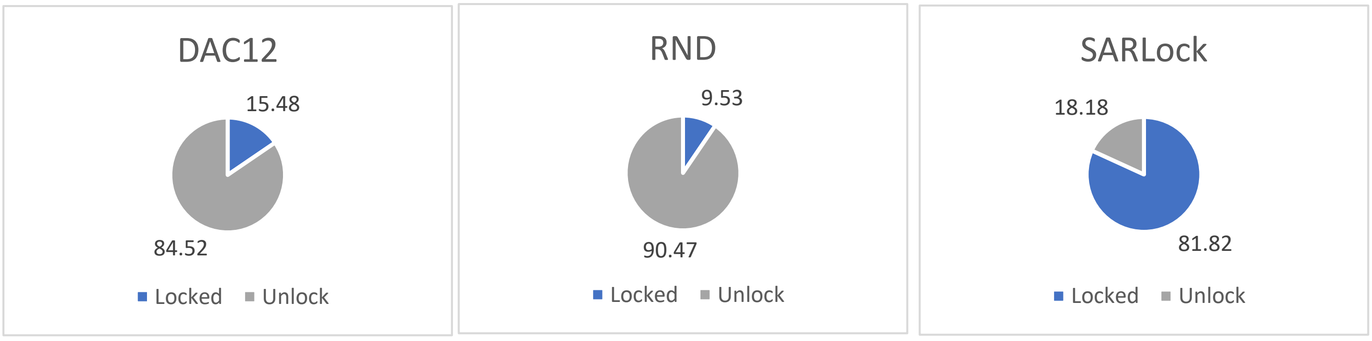


Figure 8. Pie Chart of each benchmark

Table 4 shows that the ratio of the number of unlock circuit and the number of total circuits. In our experiments result, 'dac12' and 'rnd' benchmarks are almost completely logic unlocking by SAT attack, while there are a lot of benchmarks in 'sarlock' cannot be finished in our time limitation (2hrs=7200s). By [4], we found that "SARLock" increase the required number of distinguishing input patterns exponentially with key size, by reducing the number of key values filtered in each iteration of the attack. Thus, the execution time of the attack grows exponentially with key size, and its too long to lead to TLE. Lastly, in our table, the "verification" column indicates a 100% verification rate, signifying that the key obtained from the SAT attack matches the oracle. This confirms that the SAT attack effectively solves the logic locking problem within an acceptable timeframe.

4. Conclusion

This report applied SAT attacks to locked benchmarks, including 'rnd,' 'dac12,' and 'sarlock/dac12,' with varying encryption percentages. Conducted on a CAD workstation, the experiments used Python scripts and modules for evaluation. While 'dac12' and 'rnd' benchmarks were mostly unlocked, 'sarlock' faced time limitations due to its locking skill. The verification rate confirmed successful logic unlocking. These findings contribute to secure circuit design and highlight challenges in logic locking and SAT attacks, guiding future research.

5. References:

[1] Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing (STOC '71). Association for Computing Machinery, New York, NY, USA, 151–158. https://doi.org/10.1145/800157.805047

[2] Author’s GitHub repo: https://github.com/JieHong-Liu/Advanced_Logic_Synthesis

[3] Subramanyan, P., Ray, S., & Malik, S. (2015, May). Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (pp. 137-143). IEEE.

[4] Yasin, M., Mazumdar, B., Rajendran, J. J., & Sinanoglu, O. (2016, May). SARLock: SAT attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (pp. 236-241). IEEE.

6. Appendix: All experimental results

A. DAC12:

file	inputs	keys	outputs	gates	iteration	cpu_time
dac12/apex2_enc05.bench	39	31	3	644	20	0.11469
dac12/apex2_enc10.bench	39	61	3	674	25	0.257528
dac12/apex2_enc25.bench	39	153	3	769	122	4.43968
dac12/apex2_enc50.bench	39	305	3	925	208	18.9244
dac12/apex4_enc05.bench	10	268	19	5633	90	7.56416
dac12/apex4_enc10.bench	10	536	19	5901	118	19.2294
dac12/apex4_enc25.bench	10	1340	19	6705	203	78.1807
dac12/apex4_enc50.bench	10	2680	19	8045	278	234.533
dac12/c1355_enc05.bench	41	27	32	573	2	0.20579
dac12/c1355_enc10.bench	41	55	32	601	9	0.881626
dac12/c1355_enc25.bench	41	137	32	693	29	4.48423
dac12/c1355_enc50.bench	41	273	32	837	107	82.1459
dac12/c1908_enc05.bench	33	44	25	928	29	0.383497
dac12/c1908_enc10.bench	33	88	25	976	61	1.26905
dac12/c1908_enc25.bench	33	220	25	1109	110	13.9242
dac12/c1908_enc50.bench	33	440	25	1338	146	182.374
dac12/c2670_enc05.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/c2670_enc10.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/c2670_enc25.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/c2670_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/c3540_enc05.bench	50	83	22	1754	65	2.54736
dac12/c3540_enc10.bench	50	167	22	1843	40	1.22079
dac12/c3540_enc25.bench	50	417	22	2094	71	4.16927
dac12/c3540_enc50.bench	50	835	22	2517	121	76.7445
dac12/c432_enc05.bench	36	8	7	168	1	0.021428
dac12/c432_enc10.bench	36	16	7	176	9	0.023834
dac12/c432_enc25.bench	36	40	7	200	24	0.153492
dac12/c432_enc50.bench	36	80	7	251	24	0.227551
dac12/c499_enc05.bench	41	10	32	212	5	0.039897
dac12/c499_enc10.bench	41	48	32	250	12	1.38422
dac12/c499_enc25.bench	41	51	32	253	11	1.77069
dac12/c499_enc50.bench	41	101	32	309	20	7.98892
dac12/c5315_enc05.bench	178	115	123	2424	27	7.45341

dac12/c5315_enc10.bench	178	231	123	2543	55	16.6766
dac12/c5315_enc25.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/c5315_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/c7552_enc05.bench	207	176	108	3689	88	22.4118
dac12/c7552_enc10.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/c7552_enc25.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/c7552_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/c880_enc05.bench	60	19	26	403	25	0.066089
dac12/c880_enc10.bench	60	38	26	423	21	0.179209
dac12/c880_enc25.bench	60	96	26	487	38	0.644068
dac12/c880_enc50.bench	60	192	26	584	73	2.90185
dac12/dalu_enc05.bench	75	115	16	2436	42	5.2516
dac12/dalu_enc10.bench	75	230	16	2551	43	3.60131
dac12/dalu_enc25.bench	75	575	16	2898	104	131.27
dac12/dalu_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/des_enc05.bench	256	324	245	6804	50	278.228
dac12/des_enc10.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/des_enc25.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/des_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE
dac12/ex1010_enc05.bench	10	253	10	5326	84	6.34292
dac12/ex1010_enc10.bench	10	507	10	5580	152	23.2142
dac12/ex1010_enc25.bench	10	1267	10	6340	280	94.4829
dac12/ex1010_enc50.bench	10	2533	10	7606	409	388.567
dac12/ex5_enc05.bench	8	53	63	1109	21	0.228152
dac12/ex5_enc10.bench	8	106	63	1165	29	0.413756
dac12/ex5_enc25.bench	8	264	63	1324	56	1.82416
dac12/ex5_enc50.bench	8	528	63	1588	78	6.35523
dac12/i4_enc05.bench	192	17	6	355	13	0.034185
dac12/i4_enc10.bench	192	34	6	375	49	0.247545
dac12/i4_enc25.bench	192	85	6	434	149	3.16545
dac12/i4_enc50.bench	192	129	6	488	279	15.9218
dac12/i7_enc05.bench	199	66	67	1389	14	0.171418
dac12/i7_enc10.bench	199	132	67	1470	23	0.508132
dac12/i7_enc25.bench	199	329	67	1703	26	1.05331
dac12/i7_enc50.bench	199	658	67	2043	35	2.94591
dac12/i8_enc05.bench	133	123	81	2598	31	1.18377
dac12/i8_enc10.bench	133	246	81	2721	48	2.8576

dac12/i8_enc25.bench	133	616	81	3097	73	19.6805
dac12/i8_enc50.bench	133	1232	81	3718	92	111.631
dac12/i9_enc05.bench	88	52	63	1092	17	0.313084
dac12/i9_enc10.bench	88	104	63	1150	33	1.15371
dac12/i9_enc25.bench	88	259	63	1317	34	5.02138
dac12/i9_enc50.bench	88	518	63	1576	29	5.75727
dac12/k2_enc05.bench	46	91	45	1906	62	1.15047
dac12/k2_enc10.bench	46	182	45	1997	67	2.09599
dac12/k2_enc25.bench	46	454	45	2269	96	5.69822
dac12/k2_enc50.bench	46	908	45	2723	125	30.8286
dac12/seq_enc05.bench	41	176	35	3700	73	3.55094
dac12/seq_enc10.bench	41	352	35	3879	120	9.88609
dac12/seq_enc25.bench	41	880	35	4413	213	34.8737
dac12/seq_enc50.bench	41	1760	35	5295	257	94.796

B. RND:

rnd/apex2_enc05.bench	39	31	3	643	19	0.145582	TRUE
rnd/apex2_enc10.bench	39	61	3	673	42	0.424473	TRUE
rnd/apex2_enc25.bench	39	153	3	769	134	5.15321	TRUE
rnd/apex2_enc50.bench	39	305	3	928	225	18.2128	TRUE
rnd/apex4_enc05.bench	10	268	19	5628	81	4.54754	TRUE
rnd/apex4_enc10.bench	10	536	19	5896	127	12.5784	TRUE
rnd/apex4_enc25.bench	10	1340	19	6700	201	59.072	TRUE
rnd/apex4_enc50.bench	10	2680	19	8044	276	396.917	TRUE
rnd/c1355_enc05.bench	41	27	32	574	16	0.15349	TRUE
rnd/c1355_enc10.bench	41	55	32	603	23	0.393091	TRUE
rnd/c1355_enc25.bench	41	137	32	686	66	6.04518	TRUE
rnd/c1355_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE	no need
rnd/c1908_enc05.bench	33	44	25	925	17	0.244896	TRUE
rnd/c1908_enc10.bench	33	88	25	971	28	1.37427	TRUE
rnd/c1908_enc25.bench	33	220	25	1106	63	12.9176	TRUE
rnd/c1908_enc50.bench	33	440	25	1329	96	108.474	TRUE
rnd/c2670_enc05.bench	233	60	140	1257	31	0.364789	TRUE
rnd/c2670_enc10.bench	233	119	140	1321	1761	147.468	TRUE
rnd/c2670_enc25.bench	TLE	TLE	TLE	TLE	TLE	TLE	no need

rnd/c2670_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE	no need
rnd/c3540_enc05.bench	50	83	22	1754	19	1.29351	TRUE
rnd/c3540_enc10.bench	50	167	22	1839	31	2.10607	TRUE
rnd/c3540_enc25.bench	50	417	22	2094	54	4.4193	TRUE
rnd/c3540_enc50.bench	50	835	22	2515	101	184.302	TRUE
rnd/c432_enc05.bench	36	8	7	170	2	0.02097	TRUE
rnd/c432_enc10.bench	36	16	7	179	3	0.021436	TRUE
rnd/c432_enc25.bench	36	40	7	204	15	0.050225	TRUE
rnd/c432_enc50.bench	36	80	7	248	26	0.232903	TRUE
rnd/c499_enc05.bench	41	10	32	212	3	0.046222	TRUE
rnd/c499_enc10.bench	41	20	32	224	6	0.066889	TRUE
rnd/c499_enc25.bench	41	51	32	258	15	0.406876	TRUE
rnd/c499_enc50.bench	41	101	32	313	19	2.52123	TRUE
rnd/c5315_enc05.bench	178	115	123	2427	22	0.583819	TRUE
rnd/c5315_enc10.bench	178	231	123	2548	36	1.59305	TRUE
rnd/c5315_enc25.bench	178	577	123	2905	99	22.3758	TRUE
rnd/c5315_enc50.bench	178	1154	123	3498	182	549.179	TRUE
rnd/c7552_enc05.bench	207	176	108	3695	44	2.10085	TRUE
rnd/c7552_enc10.bench	207	351	108	3877	112	16.6679	TRUE
rnd/c7552_enc25.bench	TLE	TLE	TLE	TLE	TLE	TLE	no need
rnd/c7552_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE	no need
rnd/c880_enc05.bench	60	19	26	404	4	0.035908	TRUE
rnd/c880_enc10.bench	60	38	26	423	12	0.07288	TRUE
rnd/c880_enc25.bench	60	96	26	488	16	0.266125	TRUE
rnd/c880_enc50.bench	60	192	26	590	47	2.61558	TRUE
rnd/dalu_enc05.bench	75	115	16	2418	19	0.720057	TRUE
rnd/dalu_enc10.bench	75	230	16	2533	35	2.11432	TRUE
rnd/dalu_enc25.bench	75	575	16	2882	93	19.1261	TRUE
rnd/dalu_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE	no need
rnd/des_enc05.bench	256	324	245	6804	26	1.41887	TRUE
rnd/des_enc10.bench	256	647	245	7132	39	3.4597	TRUE
rnd/des_enc25.bench	256	1618	245	8121	70	51.425	TRUE
rnd/des_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE	no need
rnd/ex1010_enc05.bench	10	253	10	5319	104	6.32855	TRUE
rnd/ex1010_enc10.bench	10	507	10	5573	155	16.5564	TRUE
rnd/ex1010_enc25.bench	10	1267	10	6333	295	101.716	TRUE
rnd/ex1010_enc50.bench	TLE	TLE	TLE	TLE	TLE	TLE	no need

rnd/ex5_enc05.bench	8	53	63	1109	14	0.157891	TRUE
rnd/ex5_enc10.bench	8	106	63	1161	35	0.444738	TRUE
rnd/ex5_enc25.bench	8	264	63	1319	42	1.096	TRUE
rnd/ex5_enc50.bench	8	528	63	1585	70	5.62259	TRUE
rnd/i4_enc05.bench	192	17	6	360	17	0.039455	TRUE
rnd/i4_enc10.bench	192	34	6	380	41	0.108	TRUE
rnd/i4_enc25.bench	192	85	6	440	68	0.405757	TRUE
rnd/i4_enc50.bench	192	169	6	534	226	8.47119	TRUE
rnd/i7_enc05.bench	199	66	67	1384	9	0.103914	TRUE
rnd/i7_enc10.bench	199	132	67	1454	11	0.137165	TRUE
rnd/i7_enc25.bench	199	329	67	1670	26	2.04704	TRUE
rnd/i7_enc50.bench	199	658	67	2017	44	2.63345	TRUE
rnd/i8_enc05.bench	133	123	81	2589	14	0.473345	TRUE
rnd/i8_enc10.bench	133	246	81	2714	20	0.839854	TRUE
rnd/i8_enc25.bench	133	616	81	3092	37	3.34897	TRUE
rnd/i8_enc50.bench	133	1232	81	3722	63	40.419	TRUE
rnd/i9_enc05.bench	88	52	63	1089	3	0.098758	TRUE
rnd/i9_enc10.bench	88	104	63	1145	10	0.211769	TRUE
rnd/i9_enc25.bench	88	259	63	1307	17	0.480665	TRUE
rnd/i9_enc50.bench	88	518	63	1574	15	1.47215	TRUE
rnd/k2_enc05.bench	46	91	45	1908	17	0.2438	TRUE
rnd/k2_enc10.bench	46	182	45	2000	32	0.572298	TRUE
rnd/k2_enc25.bench	46	454	45	2275	71	2.88284	TRUE
rnd/k2_enc50.bench	46	908	45	2731	129	44.7246	TRUE
rnd/seq_enc05.bench	41	176	35	3697	51	1.62756	TRUE
rnd/seq_enc10.bench	41	352	35	3873	72	3.58506	TRUE
rnd/seq_enc25.bench	41	880	35	4404	166	26.0651	TRUE
rnd/seq_enc50.bench	41	1760	35	5288	263	137.705	TRUE

C. SARLock:

file	inputs	keys	outputs	gates	iteration
sarlock/dac12/apex2_enc05.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/apex2_enc10.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/apex2_enc25.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/apex4_enc05.bench	10	278	19	5675	1023
sarlock/dac12/apex4_enc10.bench	10	546	19	5943	1023

sarlock/dac12/apex4_enc25.bench	10	1350	19	6747	1023
sarlock/dac12/dalu_enc05.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/dalu_enc10.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/dalu_enc25.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/des_enc05.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/des_enc10.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/des_enc25.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/ex5_enc05.bench	8	61	63	1191	255
sarlock/dac12/ex5_enc10.bench	8	114	63	1247	255
sarlock/dac12/ex5_enc25.bench	8	272	63	1406	255
sarlock/dac12/i4_enc05.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i4_enc10.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i4_enc25.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i7_enc05.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i7_enc10.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i7_enc25.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i8_enc05.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i8_enc10.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i8_enc25.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i9_enc05.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i9_enc10.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/i9_enc25.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/k2_enc05.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/k2_enc10.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/k2_enc25.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/seq_enc05.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/seq_enc10.bench	TLE	TLE	TLE	TLE	TLE
sarlock/dac12/seq_enc25.bench	TLE	TLE	TLE	TLE	TLE