Testes de aleatoriedade segundo a FIPS 140-1

Leon Bellini (22218002-8), Guilherme Ormond (222180070-7)

1 O problema

Foram fornecidas **26 chaves** em hexadecimal, as quais devem ser testadas quanto a sua possível aleatoriedade segundo os testes especificados pela **FIPS 140-1**. As informações contidas nesse relatório incluem:

1. A quantidade de *monobits*;
2. O valor calculado pelo *Poker Test*;
3. Quantidade de cada sequência de bits 0s ou 1s produzida pelo *Runs test*;
4. Uma tabela compreensível sobre a aprovação ou não de cada teste.

1.1 Algumas considerações

Para certos casos em que a conversão de hexadecimal para binário resultou em 19999 bits, foi adicionado um zero à esquerda, o que certamente influencia certos resultados.

Este relatório, a planilha de resultados, bem como a base de códigos desenvolvida pode ser encontrada no Github.

2 O procedimento

Primeiramente foram definidas funções para ajuste das strings binárias convertidas com o intuito de deixá-las todas com 20000 dígitos/bits.

```
(defn read-file
  "Receives a filename, reads a file and returns a vector of hexa strings."
  [filename]
  (with-open [reader (clojure.java.io/reader filename)]
    (reduce (fn
              [hex-vec line]
              (let [trimmed-string (clojure.string/replace (clojure.string/trim line) #"'" "")]
                (if-not (clojure.string/blank? trimmed-string)
                  (conj hex-vec trimmed-string)
                  (conj hex-vec)))) [] (line-seq reader))))

(defn transform-into-bitstrings
  "Transforms a given hexadecimal string into a string of bits."
  [hexa-vec]
  (reduce (fn
            [bit-vec hexa-string]
            (conj bit-vec (.toString (new java.math.BigInteger hexa-string 16) 2))) [] hexa-vec))

(defn adjust-strings
  "Adjusts all strings within a given vector to have 20k bits."
  [bit-vector]
  (map (fn
         [bit-string]
         (if-not (= (count bit-string) 20000)
           (str "0" bit-string)
           bit-string)) bit-vector))
```

2.1 *Monobit test*

```
(defn check-monobit
  "Checks if a given count of ones is within a pre-defined boundary."
  [one-count]
  {:one-count one-count
   :passed? (and (> one-count 9654)
                 (< one-count 10346))})

(defn monobit-test
  "Checks if a given bitstring passes the monobit test. Returns a hash."
  [bit-string]
  (loop [[curr-bit & rest-bits] bit-string one-count 0]
    (cond
      (nil? curr-bit) (check-monobit one-count)
      (= \1 curr-bit) (recur rest-bits (inc one-count))
      :else (recur rest-bits one-count))))
```

2.2 *Poker test*

```
(defn check-poker
  "Does all the logic related to the poker-test."
  [result]
  (let [sum (reduce (fn
                      [value map-t]
                      (+  (* (nth map-t 1) (nth map-t 1)) value)) 0 result)]
    (let [partial (- (* sum 16/5000) 5000)]
      {:result (float partial)
       :nibbles result
       :passed? (and (> partial 1.03 )
                     (< partial 57.4))})))

(defn poker-test
  "Checks if a string is approved by the poker test"
  [bit-string]
  (loop [lower-bound 0 upper-bound 4 results-hash {}]
    (if-not (> upper-bound (count bit-string))
      (let [temp-string (subs bit-string lower-bound upper-bound)]
        (recur (+ lower-bound 4)
               (+ upper-bound 4)
               (assoc results-hash temp-string (inc (results-hash temp-string 0)))))
      (check-poker results-hash))))
```

2.3 Long run test

```
(defn long-run-test
  [bit-string]
  {:passed? (if-not (re-find #"1{34}|0{34}" bit-string)
              true
              false)})
```

2.4 Runs test

```
(defn get-regex-count
  "Counts the number of regexp matches within a string."
  [og-string regexp]
  (count (re-seq regexp og-string)))

(defn runs?
  "Returns the quantity of runs within a certain range."
  [og-string regexp lower-bound upper-bound]
  (let [count (get-regex-count og-string regexp)]
    {:count count
     :pass? (and (> count lower-bound)
                 (< count upper-bound))}))

(defn runs-test
  "Performs the run test by applying a series of regexes."
  [bit-string]
  (let [zero-matches {:z-1 (runs? bit-string #"(?<=1)01|10$|^01" 2267 2733)
                      :z-2 (runs? bit-string #"(?<=1)0{2}1|10{2}$|^0{2}1" 1079 1421)
                      :z-3 (runs? bit-string #"(?<=1)0{3}1|10{3}$|^0{3}1" 502 748)
                      :z-4 (runs? bit-string #"(?<=1)0{4}1|10{4}$|^0{4}1" 223 402)
                      :z-5 (runs? bit-string #"(?<=1)0{5}1|10{5}$|^0{5}1" 90 223)
                      :z-6+ (runs? bit-string #"(?<=1)0{6,}1|10{6,}$|^0{6,}1" 90 223)}
        one-matches {:o-1 (runs? bit-string #"(?<=0)10|01$|^10" 2267 2733)
                     :o-2 (runs? bit-string #"(?<=0)1{2}0|01{2}$|^1{2}0" 1079 1421)
                     :o-3 (runs? bit-string #"(?<=0)1{3}0|01{3}$|^1{3}0" 502 748)
                     :o-4 (runs? bit-string #"(?<=0)1{4}0|01{4}$|^1{4}0" 223 402)
                     :o-5 (runs? bit-string #"(?<=0)1{5}0|01{5}$|^1{5}0" 90 223)
                     :o-6+ (runs? bit-string #"(?<=0)1{6,}0|01{6,}$|^1{6,}0" 90 223)}]
    {:zero-seqs zero-matches
     :one-seqs one-matches}))
```

2.5 Resultado Final

Foi obtido a partir da extração dos valores obtidos a partir da execução da seguinte função:

```
(defn test-all
  "Tests all strings and exports the final result into /tmp/resultados.txt."
  [bit-vec]
  (spit "/tmp/resultados.txt" "\n")
  (doseq [id (range 1 (inc (count bit-vec)))
          :let [single-bit-string (nth bit-vec (- id 1))]]
    (spit "/tmp/resultados.txt"
          {:id id
           :monobit (monobit-test single-bit-string)
           :long-run (long-run-test single-bit-string)}
          :append true)
    (spit "/tmp/resultados.txt" "\n" :append true)
    (spit "/tmp/resultados.txt" (poker-test single-bit-string) :append true)
    (spit "/tmp/resultados.txt" "\n" :append true)
    (spit "/tmp/resultados.txt" (runs-test single-bit-string) :append true)
    (spit "/tmp/resultados.txt" "\n" :append true)))

(defn test-strings
  []
  (let [hexa-vec (read-file "keys.txt")]
    (let [adjusted-bit-vec (adjust-strings (transform-into-bitstrings hexa-vec))]
      (test-all adjusted-bit-vec))))

(test-strings)
```

2.6 A tabela obtida

No total foram 13 chaves negadas das 26. Estas falharam em algum momento durante a bateria de testes. Deve-se notar que, para o caso do `run-test`, não houve falha em relação ao valor dos *runs* de 1.

				Runs												
				0						1						
ID	Monobit	Poker	Long-runs	1	2	3	4	5	6+	1	2	3	4	5	6+	Aprovada?
1	10052	34.2528	TRUE	2511	1246	665	323	136	137	2531	1219	627	342	144	155	Sim
2	10580	341.709	TRUE	2751	1382	719	292	88	23	2635	1319	599	362	167	172	Não
3	9944	13.1392	TRUE	2427	1274	613	322	168	164	2493	1222	632	321	146	155	Sim
4	10665	342.829	TRUE	2758	1403	702	268	85	28	2555	1325	690	335	161	178	Não
5	9984	13.1392	TRUE	2461	1209	605	336	170	166	2462	1208	634	335	159	150	Sim
6	10642	342.829	TRUE	2679	1440	702	275	85	28	2528	1314	684	336	171	176	Não
7	9883	24.4288	TRUE	2466	1223	686	324	136	167	2574	1202	645	268	155	157	Sim
8	10368	252.826	TRUE	2670	1362	726	294	109	50	2655	1279	642	318	155	163	Não
9	9853	24.4288	TRUE	2490	1241	670	331	145	157	2609	1224	620	278	156	147	Sim
10	10346	252.826	TRUE	2624	1394	762	269	108	50	2649	1321	605	308	163	161	Não
11	10761	346.925	TRUE	2758	1376	733	238	94	18	2529	1295	655	354	200	183	Não
12	9926	10.9696	TRUE	2463	1221	600	329	163	178	2478	1249	615	302	143	166	Sim
13	10677	355.45	TRUE	2863	1346	721	272	77	22	2628	1314	676	352	181	150	Não
14	10578	345.92	TRUE	2733	1410	724	263	99	25	2636	1322	612	344	161	178	Não
15	10167	17.0112	TRUE	2565	1215	602	312	152	148	2410	1298	624	340	156	165	Sim
16	10052	13.3568	FALSE	2479	1261	605	296	182	151	2474	1257	604	303	183	154	Não
17	9971	11.488	TRUE	2461	1306	630	302	158	150	2499	1294	664	322	138	140	Sim
18	10799	348.128	TRUE	2777	1417	697	237	85	21	2522	1330	651	370	180	181	Não
19	10685	343.238	TRUE	2771	1415	693	283	79	18	2553	1317	694	361	184	151	Não
20	10047	19.5392	TRUE	2477	1285	629	328	160	133	2504	1246	635	308	163	156	Sim
21	10065	19.1232	TRUE	2643	1265	614	301	155	135	2557	1324	636	323	133	141	Sim
22	10081	23.9936	TRUE	2489	1253	634	341	131	145	2435	1307	609	330	150	162	Sim
23	10564	347.936	TRUE	2726	1374	741	266	99	30	2634	1260	633	363	177	170	Não
24	9981	12.0192	TRUE	2453	1273	633	307	174	150	2507	1229	618	316	171	150	Sim
25	10593	349.715	TRUE	2601	1370	778	271	97	25	2510	1254	663	331	202	182	Não
26	10034	11.8464	TRUE	2487	1261	607	297	182	151	2479	1260	605	305	184	153	Sim

2.6.1 Os *nibbles* do *poker test*

ID	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
1	277	315	341	345	301	305	341	331	319	310	340	274	271	300	283	347
2	3	355	322	331	383	333	313	329	326	323	352	327	327	327	317	340
3	327	292	324	332	310	301	315	288	317	323	328	288	326	296	297	328
4	0	323	340	349	341	337	339	342	316	329	319	321	364	307	331	342
5	327	292	324	332	310	323	315	328	317	288	301	288	326	296	297	336
6	0	323	340	349	341	329	339	319	316	342	337	321	364	307	331	342
7	306	322	317	295	339	308	308	304	364	274	340	304	293	290	333	303
8	54	380	341	324	362	340	324	338	349	316	313	324	313	324	318	324
9	306	322	317	295	339	340	308	274	364	308	304	304	293	290	333	303
10	54	380	341	324	362	313	324	316	349	340	338	280	313	324	318	324
11	0	298	327	370	308	328	336	324	340	320	340	332	348	336	344	349
12	337	328	327	300	295	320	292	341	314	307	314	306	309	313	290	307
13	0	292	356	327	298	350	344	343	365	336	340	341	311	358	332	307
14	0	325	351	357	354	350	317	304	348	337	316	345	323	318	321	334
15	293	280	303	345	314	295	327	337	293	295	327	325	300	331	320	315
16	297	318	307	324	298	291	329	299	325	334	304	333	317	299	287	338
17	295	329	321	323	326	293	315	328	308	339	297	328	302	308	295	293
18	0	338	308	332	311	341	351	342	296	356	314	342	350	327	343	349
19	0	309	333	323	321	327	362	325	345	336	347	321	334	326	361	330
20	257	308	328	305	317	319	334	307	326	319	317	306	347	288	321	309
21	282	314	307	311	309	353	314	311	296	321	329	330	294	346	298	285
22	286	311	325	295	308	319	334	280	331	303	278	316	310	366	310	328
23	0	352	331	299	347	328	355	343	351	339	342	295	331	324	343	320
24	298	318	334	325	299	291	330	301	326	334	303	307	317	300	288	329
25	4	366	313	379	347	303	300	322	351	324	307	324	359	337	333	331
26	297	318	307	325	299	292	330	300	326	334	305	334	317	300	288	328