Strong stationary times and its use in cryptography

Paweł Lorek, Filip Zagórski, and Michał Kulis

Abstract—This paper presents applicability of Strong Stationary Times (SST) techniques in the area of cryptography. The applicability is in three areas:

1) Propositions of a new class of cryptographic algorithms (pseudo-random permutation generators) which do not run for the predefined number of steps. Instead, these algorithms stop according to a stopping rule defined as SST, for which one can obtain provable properties:
   a) results are perfect samples from uniform distribution,
   b) immunity to timing attacks (no information about the resulting permutation leaks through the information about the number of steps SST algorithm performed).
2) We show how one can leverage properties of SST-based algorithms to construct an implementation (of a symmetric encryption scheme) which is immune to the timing-attack by reusing implementations which are not secure against timing-attacks. In symmetric key cryptography researchers mainly focus on constant time (re)implementations. Our approach goes in a different direction and explores ideas of input masking.
3) Analysis of idealized (mathematical) models of existing cryptographic schemes – *i.e.,* we improve a result by Mironov [21].

Index Terms—Pseudo-random permutation generator, Markov chains, mixing time, stream cipher, timing attacks

1 Introduction

In the traditional cryptography primitives are treated as mathematical objects, security definitions involve *black-box* model: it is assumed that an adversary has only access to inputs and outputs of a given cryptographic scheme. In reality however, a device, or a particular implementation, may expose lots of additional side-channel information, *the running time* being one of the most important. It can be particularly dangerous, since one can often measure the time even without physical access to the device. Attacks exploiting this are called *timing attacks*.

Algorithm 1 Simple modular exponentiation $c^d \bmod n$

Require: $c, d = d_s d_{s-1} \cdots d_1 d_0; d_i \in \{0,1\}$
1: $x = 1$
2: **for** i = s **to** 0 **do**
3: $x = mod(x \cdot x, n)$
4: **if** $d_i == 1$ **then**
5: $x = mod(x \cdot c, n)$
6: **end if**
7: **end for**
8: **return** x

- *M. Kulis and F. Zagórski are with the Department of Computer Science Faculty of Fundamental Problems of Technology, Wrocław University of Science and Technology, Poland.*
- *P. Lorek is with the Mathematical Institute, Faculty of Mathematics and Computer Science, Wrocław University, Poland. Email Pawel.Lorek@math.uni.wroc.pl*

All authors were supported by Polish National Science Centre contract number DEC-2013/10/E/ST1/00359.

Straightforward implementations of RSA used the modular exponentiation algorithm as described in Algorithm 1 together with Algorithm 2.

Algorithm 2 Modular division $mod(x, n) = x \bmod n$

Require: x, n
1: **if** $x \geq n$ **then**
2: $x = x\%n$
3: **end if**
4: **return** x

An adversary, by careful selection of messages c, and just by measuring the time it takes to compute the result is able to completely recover d – the private key (since the number of times line number 2 of the Algorithm 2 and line number 5 of the Algorithm 1 are called depends both on c and d). Kocher [16] was the first one to discuss timing attacks.

How can one prevent timing attacks? There are two natural approaches:

A1) **Constant-time implementation**. Re-implementing the algorithm in such a way that the running time is data independent (it runs in the same time on all possible inputs).

A2) **Masking (blinding)**. Use some extra randomness in such a way that time-dependent operations are performed on random inputs.

The A1 approach is self-explaining. However, not all algorithms are easily transformable to constant time ones. Moreover, the compiler in some cases can optimize the algorithm resulting in data-dependent running time again,

or a constant-time implementation on one platform may become again time attack susceptible on a different platform – see [23], [26].

In this paper we focus our attention on masking (*i.e.,* A2) by performing some "random" permutation first, which is key-dependent.

Authors of [7] presented fast constant-time algorithms for code-based public-key cryptography called *McBits*. McBits uses a permutation network to achieve immunity against cache-timing attacks. Permutation network is a network built up of wires and comparators. Wires carry values while comparators sort values between two wires (smaller value goes to the upper wires, bigger value goes to the lower wires). Permutation network is a fixed construction: number of wires and comparators is fixed. When such a network has the property that it sorts all inputs, then it is called a sorting network.

In Section 5 of [7] on "Secret permutations" authors write (where a permutation is considered as the output of a permutation network)

> *"By definition a permutation network can reach every permutation, but perhaps it is much more likely to reach some permutations than others. Perhaps this hurts security. Perhaps not [...]"*

Above motivation leads to the following question:

> *How to obtain a key-based permutation which is indistinguishable from a truly random one?*

The doubt about the resulting permutation in [7] can come from the following: one can have a recipe for constructing a permutation involving some number of single steps, determining however the necessary number of steps is not trivial. In this paper we resolve the issue, our considerations are Markov chain based. In particular, we consider some card shuffling schemes. Our methods can have two applications: we can determine number of steps so that we are arbitrary close to uniform permutation; in so-called SST version algorithm does not run pre-defined number of steps, but instead it stops once randomness is reached. As a side-effect we have a detailed analysis of some existing stream ciphers. Many stream ciphers (e.g., RC4, RC4A [30], RC4+ [29], VMPC [33], Spritz [28]) are composed of two algorithms:

1) KSA (Key Scheduling Algorithm) uses a secret key to transform identity permutation of n elements into some other permutation.
2) PRGA (Pseudo Random Generation Algorithm) starts with a permutation generated by KSA and outputs random bits from it (some function of the permutation and the current step number) while updating the permutation at the same time.

KSAs of all aforementioned algorithms (RC4, RC4A, Spritz, RC4+, VMPC) can be seen as performing some card shuffling, where a secret key corresponds to/replaces randomness. If we consider a version of the algorithm with purely random secret key of infinite length, then we indeed consider a card shuffling procedure. Following [21], we will refer to such a version of the algorithm as an *idealized version*. In the case of KSA used by RC4 the idealized version (mathematical model) of the card shuffle is called *Cyclic-to-Random Transpositions* shuffle.

The KSAs of mentioned ciphers perform shuffling for some **predefined number of steps**. The security of such a scheme is mainly based on analyzing idealized version of the algorithm and corresponds to the "quality of a shuffling". Roughly speaking, shuffling is considered as a Markov chain on permutations, all of them converge to uniform distribution (perfectly shuffled cards). Then we should perform as many steps as needed to be close to this uniform distribution, what is directly related to the so-called *mixing time*. This is one of the main drawbacks of RC4: it performs *Cyclic-to-Random Transpositions* for n steps, whereas the mixing time is of the order $n \log n$.

There is a long list of papers which point out weaknesses of the RC4 algorithm. Attacks exploited both weaknesses of PRGA and KSA or the way RC4 was used in specific systems [5], [13], [14], [15], [19]. As a result, in 2015 RC4 was prohibited in TLS by IETF, Microsoft and Mozilla.

In the paper we use so-called **Strong Stationary Times** (SST) for Markov chains. The main area of application of SSTs is studying the rate of convergence of a chain to its stationary distribution. However, they may also be used for *perfect sampling* from stationary distribution of a Markov chain, consult [27] (on Coupling From The Past algorithm) and [12] (on algorithms involving Strong Stationary Times and Strong Stationary Duality).

This paper is an extension (and applications to timing attacks) of [17] presented at Mycrypt 2016 conference.

1.1 Related work - timing attacks

Kocher [16] was first to observe that information about the time it takes a given cryptographic implementation to compute the result may leak information about the secret (or private) key. Timing attack may be treated as the most dangerous kind of side-channel attacks since it may be exploited even by a remote adversary while most of other types of side-channel attacks are distance bounded. Kocher discussed presented timing attacks on RSA, DSS and on symmetric key encryption schemes.

Similar timing attack was implemented for CASCADE smart cards [10]. Authors report recovering 512-bit key after 350.000 timing measurements (few minutes in late 90s). Later, Brumley and Boneh [8] showed that timing attacks may be performed remotely and on algorithms that were previously not considered vulnerable (RSA using Chinese Remainder Theorem).

The usual way of dealing with timing attacks is to attempt to re-implement a given scheme in such a way that it runs the same number of steps (*i.e.,* approach **A1**), no matter what the input is. This approach has the following drawbacks:

1) it requires to replace cryptographic hardware,
2) it is hard to do it right,
3) it badly influences the performance.

(1) The first limitation is quite obvious. So it may be worth considering the idea of reusing a timing attack susceptible implementation by masking (see Section 1.1.1) its inputs. This was the original timing attack countermeasure technique proposed by Kocher for both RSA and DSS.

(2) An example of s2n – Amazon's TLS implementation and its rough road to become immune to timing attacks [2]

shows that achieving timing resistance is not easy. It is not an isolated case, it may be hard to achieve constant time in the case of of implementation [24] but what may be more worrying, even a change at the architecture level may not lead to success [31].

(3) BearSSL is an implementation of TLS protocol which consists of many constant time implementations (along with a "traditional" ones) of the main cryptographic primitives. Constant-time implementation of CBC encryption for AES is 5.89 times slower than the "standard" implementation [25]. On a reference platform it achieves the following performance: AES-CBC (encryption, big): 161.40 MB/s while the constant time (optimized for 64-bit architecture) has 27.39 MB/s. The slow-down is not always at the same rate, it depends on both the function and on the mode of operation used (for AES in CTR mode the slowdown is 1.86; for 3DES it is 3.10).

1.1.1 Masking

Let $F(\cdot)$ be the RSA decryption (described as Algorithm 1) implementation which uses a private key $[d, N]$ to decrypt an encrypted message $x^e \bmod N$. The following shows how to design $\mathcal{F}$ which is immune to timing attacks but reuses implementation of F. The approach proposed by Kocher [16] is to run a bad implementation F on a random input (same idea as Chaum's blind-signatures [9]), the code for $\mathcal{F}$ is presented in Algorithm 3.

Algorithm 3 $\mathcal{F}$ – timing attack immune implementation of RSA decryption using time-attack susceptible implementation F

Require: x, F, d, e, N
1: select r uniformly at random from Z_N^*
2: compute $y = F(d, xr^e \bmod N)$
3: **return** $s = yr^{-1} \bmod N$

The security of this approach comes from the fact that while the running time of G is not constant, an attacker cannot deduce information about the key, because it does not know the input to the F. This is because $xr^e \bmod N$ is a random number uniformly distributed in Z_N^*.

The beauty of this approach lies in the fact that F is actually run on a random input. After applying F one can easily get rid of it. To be more precise: let $g_1(x, r) = xr^e \bmod N$ and let $g_2(x, r) = xr^{-1} \bmod N$ then $G(d, x) = g_2(F(d, g_1(x, r)), r) = F(d, x)$ (both g_1 and g_2 depend on the public key $[N, e]$). Taking advantage of properties of the underlying group lets us to remove the randomness r (which remains secret to the attacker during the computations).

Similar approach can be applied to other public key schemes *e.g.*, RSA signatures, DSA signatures. It is unknown how this approach – how to design G – may be applied to symmetric-key cryptographic schemes without introducing any changes to the original function F.

1.2 Our contribution

(1) Strong stationary time based PRNGs (Pseudo Random Number Generators). Instead of running a PRNG algorithm for some pre-defined number of steps, we make it randomized (Las Vegas algorithm). As far as we are aware the idea of key-dependent running time in context of symmetric ciphers was not considered earlier. To be more specific we suggest utilization of so-called **Strong Stationary Times** (SST) for Markov chains. We use SST to obtain samples from uniform distribution on all permutations (we actually perform perfect sampling). Few things needs pointing out:

1) In McBits [7] construction, authors perform *similar* masking by performing some permutation. This is achieved via a permutation network. However, they: a) run it for a pre-defined number of steps (they are unsure if the resulting permutation is random); b) they work hard to make the implementation constant-time.
   Utilization of SST allows to obtain a *perfect* sample from uniform distribution (once SST rule is found, we have it "for free"). This is a general concept which may be applied to many algorithms.
2) Coupling methods are most commonly used tool for studying the rate of convergence to stationarity for Markov chains. They allow to bound the so-called *total variation distance* between the distribution of given chain at time instant k and its stationary distribution. However, the (traditional) security definitions require something "stronger". It turns out that bounding *separation distance* is what one actually needs. It fits perfectly into the notion of Strong Stationary Times we are using (see Section 2.3).
3) By construction, the running time of our model is key dependent. In extreme cases (very unlikely) it may leak some information about the key, but it **does not leak any** information about the resulting permutation to an adversary. We elaborate more on this in Section 6.5.

(2) Preventing timing attacks. We present a technique which allows to obtain a timing attack immune implementation of a symmetric key encryption scheme by re-using a timing attack vulnerable scheme. The technique follows ideas of Kocher's masking of the RSA algorithm and are similar in nature to the utilization of a permutation network in McBits.

The idea is that an input to a timing-attack vulnerable function is first a subject to a pseudo-random permutation. The permutation is generated (from a secret key) via SST-based pseudo-random permutation generation algorithm.

The main properties of such an approach are following: the resulting permutation is uniformly random provided the long enough key is applied; the running time of an SST-based algorithm does not reveal any information about the resulting permutation.

Based on our straightforward implementation, we show that the performance of our approach is similar to constant-time implementations available in BearSSL [25], see Section 7, Figure 7 (but does not require that much effort and caution to implement it right).

(3) Analysis of idealized models of existing cryptographic schemes.

The approach, in principle, allows us to analyze the quality of existing PRNGs, closing the gap between theoretical models and practice. The "only" thing which must be done for a given scheme is to find an SST for the corresponding shuffle and to analyze its properties. We present in-depth analysis of RC4's KSA algorithm. As a result of Mironov's [21] work, one knows that the idealized version of RC4's KSA would need keys of length $\approx 23\ 037$ in order to meet the mathematical model. Similarly as in [21], we propose SST which is valid for *Cyclic-to-Random Transpositions* and *Random-to-Random Transpositions*. For the latter one we calculate the mixing time which is "faster". It is known that *Random-to-Random Transpositions* card shuffling needs $\frac{1}{2}n\log n$ steps to mix. It is worth mentioning that this shuffling scheme and *Cyclic-to-Random Transpositions* are similar in the *spirit*, it does not transfer automatically that the latter one also needs $\frac{1}{2}n\log n$ steps to mix. Mironov [21] states that the expected time till reaching uniform distribution is upper bounded by $2nH_n - n$, we show that the expected running time for this SST applied to *Random-to-Random Transpositions* is equal to

$$E[T] = nH_n + n + O(H_n)$$

(H_n is the n-th harmonic number) and empirically check that the result is similar for *Cyclic-to-Random Transpositions*. This directly translates into the required steps that should be performed by RC4.

In fact one may use a better shuffling than the one that is used in RC4, *e.g., time-reversed riffle shuffle* which requires (on average) 4096 bits – not much more than 2048 bits which are allowed for RC4 (see Section 5).

2 Preliminary

Throughout the paper, let $\mathcal{S}_n$ denote a set of all permutations of a set $\{1,\ldots,n\} =: [n]$.

2.1 Markov chains and rate of convergence

Consider an ergodic Markov chain $\mathbf{X} = \{X_k, k \geq 0\}$ on a finite state space $\mathbb{E} = \{0,\ldots,M-1\}$ with a stationary distribution ψ. Let $\mathcal{L}(X_k)$ denote the distribution of the chain at time instant k. By the *rate of convergence* we understand the knowledge on how fast the distribution of the chain converges to its stationary distribution. We have to measure it according to some distance $dist$. Define

$$\tau_{mix}^{dist}(\varepsilon) = \inf\{k : dist(\mathcal{L}(X_k), \psi) \leq \varepsilon\},$$

which is called *mixing time* (w.r.t. given distance $dist$). In our case the state space is the set of all permutations of $[n]$, *i.e.,* $\mathbb{E} := \mathcal{S}_n$. The stationary distribution is the uniform distribution over $\mathbb{E}$, *i.e.,* $\psi(\sigma) = \frac{1}{n!}$ for all $\sigma \in \mathbb{E}$. In most applications the mixing time is defined w.r.t. total variation distance:

$$d_{TV}(\mathcal{L}(X_k), \psi) = \frac{1}{2}\sum_{\sigma\in\mathcal{S}_n}\left|Pr(X_k = \sigma) - \frac{1}{n!}\right|.$$

The **separation distance** is defined by:

$$sep(\mathcal{L}(X_k), \psi) := \max_{\sigma\in\mathbb{E}}\left(1 - n!\cdot Pr(X_k = \sigma)\right).$$

It is relatively easy to check that $d_{TV}(\mathcal{L}(X_k), \psi) \leq sep(\mathcal{L}(X_k), \psi)$.

Strong Stationary Times. The definition of separation distance fits perfectly into the notion of Strong Stationary Times (SST) for Markov chains. This is a probabilistic tool for studying the rate of convergence of Markov chains allowing also *perfect sampling*. We can think of stopping time as of a running time of an algorithm which observes a Markov chain $\mathbf{X}$ and which stops according to some stopping rule (depending only on the past).

Definition 2.1. Random variable T is a randomized stopping time if it is a running time of the RANDOMIZED STOPPING TIME algorithm.

Algorithm 4 RANDOMIZED STOPPING TIME

1: $k := 0$
2: $coin$:=`Tail`
3: **while** $coin$ ==`Tail` **do**
4: At time k, $(X_0,\ldots,X_k)$ was observed
5: Calculate $f_k(X)$, where $f_k : (X_0,\ldots,X_k) \to [0,1]$
6: Let $p = f_k(X_0,\ldots,X_k)$. Flip the coin resulting in `Head` with probability p and in `Tail` with probability $1-p$. Save result as $coin$.
7: $k := k+1$
8: **end while**

Definition 2.2. Random variable T is a **Strong Stationary Time (SST)** if it is a randomized stopping time for chain $\mathbf{X}$ such that:

$$\forall(i \in \mathbb{E})\ Pr(X_k = i | T = k) = \psi(i).$$

Having SST T lets us bound the following (see [4])

$$sep(\mathcal{L}(X_k), \psi) \leq Pr(T > k). \tag{1}$$

2.2 Distinguishers and security definition

In this section we define a Permutation distinguisher – we consider its advantage in a traditional cryptographic security definition. That is, a Permutation distinguisher is given a permutation π and needs to decide whether π is a result of a given (pseudo-random) algorithm or if π is a permutation selected uniformly at random from the set of all permutations of a given size. The upper bound on Permutation distinguisher's advantage is an upper bound on any possible distinguisher – even those which are not bounded computationally. Additionally, in the Section 4.1 we consider Sign distinguisher.

Permutation distinguisher The distinguishability game for the adversary is as follows (Algorithm 5):

Definition 2.3. The permutation indistinguishability $\mathsf{Shuffle}_{\mathsf{S},\mathcal{A}}(n,r)$ experiment:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2017.2751475, IEEE Transactions on Dependable and Secure Computing

Algorithm 5 $\mathsf{Shuffle}_{\mathsf{S},\mathcal{A}}(n,r)$

Let S be a shuffling algorithm which in each round requires m bits, and let $\mathcal{A}$ be an adversary.

1) S is initialized with:
   a) a key generated uniformly at random $\mathcal{K} \sim \mathcal{U}(\{0,1\}^{rm})$,
   b) $S_0 = \pi_0$ (identity permutation)
2) S is run for r rounds: $S_r := \mathsf{S}(\mathcal{K})$ and produces a permutation π_r.
3) We set:
   - $c_0 := \pi_{rand}$ a random permutation from uniform distribution is chosen,
   - $c_1 := \pi_r$.
4) A challenge bit $b \in \{0,1\}$ is chosen at random, permutation c_b is sent to the Adversary.
5) Adversary replies with b'.
6) The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

In the case when the adversary wins the game (if $b = b'$) we say that $\mathcal{A}$ succeeded. The adversary wins the game if it can distinguish the random permutation from the permutation being a result of the PRPG algorithm.

Definition 2.4. A shuffling algorithm S generates indistinguishable permutations if for all adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$Pr\left[\mathsf{Shuffle}_{\mathsf{S},\mathcal{A}}(n,r) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n).$$

The above translates into:

Definition 2.5. A shuffling algorithm S generates indistinguishable permutations if for any adversary $\mathcal{A}$ there exists a negligible function negl such that:

$$\left| \Pr_{K \leftarrow \{0,1\}^{keyLen}} [\mathcal{A}(\mathsf{S}(K)) = 1] - \Pr_{R \leftarrow \mathcal{U}(\mathcal{S}_n)} [\mathcal{A}(R) = 1] \right| \leq \mathsf{negl}(n).$$

2.3 Strong stationary times and security guarantees

Coupling method is a commonly used tool for bounding the rate of convergence of Markov chains. Roughly speaking, a coupling of a Markov chain $\mathbf{X}$ with a transition matrix $\mathbf{P}$ is a bivariate chain $(\mathbf{X}', \mathbf{X}'')$ such that marginally $\mathbf{X}'$ and $\mathbf{X}''$ are Markov chains with the transition matrix $\mathbf{P}$ and once the chains meet they stay together (in some definitions this condition can be relaxed). Let then $T_c = \inf_k\{X'_k = X''_k\}$ *i.e.,* the first time chains meet, called *coupling time*. The *coupling inequality* states that $d_{TV}(\mathcal{L}(X_k), \psi) \leq Pr(T_c > k)$.

On the other hand, separation distance is an upper bound on total variation distance, *i.e.,* $d_{TV}(\mathcal{L}(X_k), \psi) \leq sep(\mathcal{L}(X_k), \psi)$. At first glance it seems that it is better to directly bound d_{TV}, since we can have d_{TV} very small, whereas sep is (still) large. However, knowing that sep is small gives us much more than just knowing that d_{TV} is small, what turns out to be **crucial** for proving security guarantees (*i.e.,* Definition 2.5). In our case ($\mathbb{E} = \mathcal{S}_n$ and ψ is a uniform distribution on $\mathbb{E}$) having d_{TV} small, *i.e.,* $d_{TV}(\mathcal{L}(X_k), \psi) = \frac{1}{2}\sum_{\sigma \in \mathcal{S}_n} \left|Pr(X_k = \sigma) - \frac{1}{n!}\right| \leq \varepsilon$ **does not** imply that $|Pr(X_k = \sigma) - \frac{1}{n!}|$ is uniformly small (*i.e.,* of order $\frac{1}{n!}$). Knowing however that $sep(\mathcal{L}(X_k), \psi) \leq \varepsilon$ implies

$$\forall(\sigma \in \mathbb{E}) \ \left|Pr(X_k = \sigma) - \frac{1}{n!}\right| \leq \frac{\varepsilon}{n!}. \tag{2}$$

Above inequality is what we need in our security definitions and shows that the notion of separation distance is an adequate measure of mixing time for our applications.

It is worth noting that $d_{TV}(\mathcal{L}(X_k), \mathcal{U}(\mathbb{E})) \leq \varepsilon$ implies (see Theorem 7 in [3]) that $sep(\mathcal{L}(X_{2k}), \mathcal{U}(\mathbb{E})) \leq \varepsilon$. This means that proof of security which bounds directly total variation distance by ε would require twice as many bits of randomness compared to the result which guarantees ε bound on the separation distance.

3 Shuffling cards

In this Section we consider four chains $\mathbf{X}^{(T2R)}, \mathbf{X}^{(RTRT)}, \mathbf{X}^{(CTRT)}, \mathbf{X}^{(RS)}$ corresponding to known shuffling schemes: *Top-To-Random*, *Random-To-Random Transpositions*, *Cyclic-To-Random Transpositions* and *Riffle Shuffle* (we actually consider its time reversal). The state space of all chains is $\mathbb{E} = \mathcal{S}_n$. The generic state is denoted by $S \in \mathbb{E}$, where $S[j]$ denotes the position of element j. Rough description of one step of each shuffling (in each shuffling "random" corresponds to the uniform distribution) is following:

- $\mathbf{X}^{(T2R)}$: choose random position j and move the card $S[1]$ (i.e., which is currently on top) to this position.
- $\mathbf{X}^{(RTRT)}$: choose two positions i and j at random and swap cards $S[i]$ and $S[j]$.
- $\mathbf{X}^{(CTRT)}$: at step t choose random position j and swap cards $S[(t \bmod n) + 1]$ and $S[j]$.
- $\mathbf{X}^{(RS)}$: assign each card random bit. Put all cards with bit assigned 0 to the top keeping relative positions of these cards.

The stationary distribution of all the chains is uniform, i.e. $\psi(\sigma) = \frac{1}{n!}$ for all $\sigma \in \mathbb{E}$. Aforementioned Strong Stationary Time is one of the methods to study rate of convergence. Probably the simplest (in description) is an SST for $\mathbf{X}^{(T2R)}$:

> Wait until the card which is initially at the bottom reaches the top, then make one step more.

Denote this time by $T^{(T2R)}$. In this classical example $T^{(T2R)}$ corresponds to the classical coupon collector problem. We know that $Pr(T^{(T2R)} > n\log n + cn) \leq e^{-c}$, thus via (1) we have

$$d_{TV}(\mathcal{L}(X_k^{(T2R)}), \psi) \leq sep(\mathcal{L}(X_k^{(T2R)}), \psi) \leq e^{-c}$$

for $k = n\log n + cn$. This bound is actually tight, in a sense that there is a so-called cutoff at $n\log n$, see Aldous and Diaconis [3].

The chain $\mathbf{X}^{(RTRT)}$ mixes at $\frac{1}{2}n\log n$, the result was first proven by Diaconis and Shahshahani [11] with analytical methods, Matthews [20] provided construction of SST for this chain.

There is a simple construction of SST for $\mathbf{X}^{(RS)}$: initially consider all pairs of cards as *unmarked*. Then, if at some step cards in some pair were assigned different bits *mark* this pair. Stop at the first time when all pairs are marked. This SST yields in $2n\log_2 n$ mixing time (the actual answer is that $(3/2)n\log_2 n$ steps are enough).

The case $\mathbf{X}^{(CTRT)}$ is a little bit more complicated. Having SST T does not automatically imply that we are able to bound $Pr(T > k)$, or even calculate ET. This is due to the cyclic structure of the shuffle. Note that *"in spirit"* $\mathbf{X}^{(CTRT)}$ is similar to $\mathbf{X}^{(RTRT)}$, which - as mentioned - mixes at $\frac{1}{2}n\log n$ (and even more - there is a so-called cutoff at this value, meaning that "most of the action happens" around $\frac{1}{2}n\log n$). Mironov [21] (in context of analysis of RC4) showed that it takes $O(n\log n)$ for this shuffle to mix. Unfortunately Mironov's "estimate of the rate of growth of the strong stationary time T is quite loose" and results "are a far cry both from the provable upper and lower bounds on the convergence rate". He:

- proved an upper bound $O(n\log n)$. More precisely Mironov showed that there exists some positive constant c such that $P[T > cn\log n] \to 0$ when $n \to \infty$. Author experimentally checked that $P[T > 2n\lg n] < 1/n$ for $n = 256$ which corresponds to $P[T > 4096] < 1/256$.
- experimentally showed that $E[T] \approx 11.16n \approx 1.4n\lg n \approx 2857$ (for $n = 256$) – which translates into: on average one needs to drop ≈ 2601 initial bytes.

Later Mosel, Peres and Sinclair [22] proved a matching lower bound establishing mixing time to be of order $\Theta(n\log n)$. However, the constant was not determined.

One can apply the following approach: find T which is a proper SST for *both* $\mathbf{X}^{(CTRT)}$ and $\mathbf{X}^{(RTRT)}$, calculate its expectation for $\mathbf{X}^{(RTRT)}$ and show experimentally that the expectation is similar for $\mathbf{X}^{(CTRT)}$. This is the main part of this Section resulting in a conjecture that the mixing time of $\mathbf{X}^{(CTRT)}$ is $n\log n$.

3.1 Mironov's SST for *Cyclic-to-Random Transpositions*

Let us recall Mironov's [21] construction of SST:

> "At the beginning all cards numbered $0,\ldots,n-2$ are *unchecked*, the $(n-1)^{th}$ card is *checked*. Whenever the shuffling algorithm exchanges two cards, $S[i]$ and $S[j]$, one of the two rules may apply before the swap takes place:
>
> a. If $S[i]$ is unchecked and $i = j$, check $S[i]$.
> b. If $S[i]$ is unchecked and $S[j]$ is checked, check $S[i]$.
>
> The event T happens when all cards become checked. "

Then the author proves that this is a SST for *Cyclic-to-Random Transpositions* and shows that there exists constant c (can be chosen less than 30) such that $Pr[T > cn\log n] \to 0$ when $n \to \infty$. Note that this marking scheme is also valid for *Random-to-Random Transpositions* shuffling, for which we have:

Lemma 1. The expected running time of *Random-to-Random Transpositions* shuffling with Mironov stopping rule is:

$$ET = 2nH_n - n + O(H_n).$$

Proof. We start with one card checked. When k cards are checked, then probability of checking another one is equal to $p_k = \frac{(n-k)(k+1)}{n^2}$. Thus, the time to check all the cards is distributed as a sum of geometric random variables and its expectation is equal to:

$$\sum_{k=1}^{n-1}\frac{1}{p_k} = 2\frac{n^2}{n+1}H_n - n = 2nH_n - n + O(H_n).$$

□

3.2 Better SST for *Cyclic-to-Random Transpositions*

We suggest another "faster" SST which is valid for both *Cyclic-to-Random Transpositions* and *Random-to-Random Transpositions*. We will calculate its expectation and variance for *Random-to-Random Transpositions* and check experimentally that it is similar if the stopping rule is applied to *Cyclic-to-Random Transpositions*.
The SST is given in `StoppingRuleKLZ` algorithm.

Algorithm 6 `StoppingRuleKLZ`

Input set of already marked cards $\mathcal{M} \subseteq \{1,\ldots,n\}$, round r, $Bits$
Output {`YES`, `NO`}

$j =$`n-value`$(Bits)$
if there are less than $\lceil (n-1)/2 \rceil$ marked cards **then**
  if both $\pi[r]$ and $\pi[j]$ are unmarked **then**
    mark card $\pi[r]$
  end if
else
  if ($\pi[r]$ is unmarked and $\pi[j]$ is marked) OR ($\pi[r]$ is unmarked and $r = j$) **then**
    mark card $\pi[r]$
  end if
end if

if all cards are marked **then**
  `STOP`
else
  `CONTINUE`
end if

Lemma 2. The stopping rule `StoppingRuleKLZ` is SST for $\mathbf{X}^{(CTRT)}$.

Proof. First phase of the procedure (*i.e.*, the case when there are less than $\lceil (n-1)/2 \rceil$ cards marked) is a construction a random permutation of marked cards by placing unmarked cards on randomly chosen unoccupied positions – this is actually the first part of Matthews's [20] marking scheme. Second phase is simply a Broder's construction. Theorem 9. in [21] shows that this is a valid SST for *Cyclic-to-Random Transpositions*. Both phases combined produce a random permutation of all cards. □

Remark 1. One important remark should be pointed. Full Matthews's marking [20] scheme is an SST which is "faster" than ours. However, although it is valid for *Random-to-Random Transpositions*, it is not valid for *Cyclic-to-Random Transpositions*.

Calculating ET or $VarT$ seems to be a challenging task. But note that marking scheme `StoppingRuleKLZ` also yields a valid SST for *Random-to-Random Transpositions*. In next Lemma we calculate ET and $VarT$ for this shuffle, later we experimentally show that ET is very similar for both marking schemes.

Lemma 3. Let T be the SST `StoppingRuleKLZ` applied to *Random-to-Random Transpositions*. Then we have

$$\begin{aligned} E[T] &= nH_n + n + O(H_n), \\ Var[T] &\sim \tfrac{\pi^2}{4}n^2, \end{aligned} \tag{3}$$

where $f(k) \sim g(k)$ means that $\lim_{k\to\infty} \frac{f(k)}{g(k)} = 1$.

Proof.

Define T_k to be the first time when k cards are marked (thus $T \equiv T_n$). Let $d = \lceil (n-1)/2 \rceil$. Then T_d is the running time of the first phase and $(T_n - T_d)$ is the running time of the second phase. Denote $Y_k := T_{k+1} - T_k$.

Assume that there are $k < d$ marked cards at a certain step. Then the new card will be marked in next step if we choose two unmarked cards what happens with probability:

$$p_a(k) = \frac{(n-k)^2}{n^2}.$$

Thus Y_k is a geometric random variable with parameter $p_a(k)$ and

$$\begin{aligned} &E[T_d] = \\ &= \sum_{k=0}^{d-1} E[Y_k] = \sum_{k=0}^{d-1} \frac{1}{p_a(k)} = \sum_{k=0}^{d-1} \frac{n^2}{(n-k)^2} = n^2 \sum_{k=n-d+1}^{n} \frac{1}{k^2} \\ &= n^2 \left(H_n^{(2)} - H_{n-d}^{(2)} \right) = n^2 \left(\frac{1}{n} + O\left(\frac{1}{n^2} \right) \right) = n + O(1). \end{aligned}$$

Now assume that there are $k \geq d$ cards marked at a certain step. Then, the new card will be marked in next step with probability:

$$p_b(k) = \frac{(n-k)(k+1)}{n^2}$$

and Y_k is a geometric random variable with parameter $p_a(k)$. Thus:

$$\begin{aligned} &E[T_n - T_d] \\ &= \textstyle\sum_{k=0}^{d-1} E[Y_k] = \sum_{k=d}^{n-1} \frac{1}{p_b(k)} = \sum_{k=d}^{n-1} \frac{n^2}{(n-k)(k+1)} \\ &= \tfrac{n^2}{n+1} \textstyle\sum_{k=d}^{n-1} \left(\frac{1}{n-k} + \frac{1}{k+1} \right) = \frac{n^2}{n+1} \left(\sum_{k=1}^{n-d} \frac{1}{k} + \sum_{k=d+1}^{n} \frac{1}{k} \right) \\ &= \tfrac{n^2}{n+1} \left(H_{n-d} + H_n - H_d \right) = \tfrac{n^2}{n+1} H_n + \tfrac{n^2}{n+1} \left(H_{n-d} - H_d \right) \\ &= nH_n - \tfrac{n}{n+1} H_n + \tfrac{n^2}{n+1} \left(H_{n-d} - H_d \right) \\ &= nH_n + O(H_n) + O(1) = nH_n + O(H_n). \end{aligned}$$

For variance we have:

$$\begin{aligned} &Var[T_d] = \\ &\sum_{k=0}^{d-1} \frac{1 - p_a(k)}{\left(p_a(k)\right)^2} = \sum_{k=0}^{d-1} \frac{1 - \frac{(n-k)^2}{n^2}}{\left(\frac{(n-k)^2}{n^2}\right)^2} \approx \int_0^{\frac{n}{2}} \frac{1 - \frac{(n-x)^2}{n^2}}{\left(\frac{(n-x)^2}{n^2}\right)^2} dx = \frac{4}{3}n. \\ &Var[T_n - T_d] = \\ &\sum_{k=d}^{n-1} Var[Y_k] = \sum_{k=d}^{n-1} \frac{1 - p_b(k)}{\left(p_b(k)\right)^2} = \sum_{k=d}^{n-1} \frac{1 - \frac{(n-k)(k+1)}{n^2}}{\frac{(n-k)^2(k+1)^2}{n^4}} \\ &= n^2 \sum_{k=d}^{n-1} \frac{n(n-1) + k(1-n) + k^2}{(n-k)^2(k+1)^2} \\ &\approx n^2 \cdot \frac{1}{2} \sum_{k=0}^{n-1} \frac{n(n-1) + k(1-n) + k^2}{(n-k)^2(k+1)^2} \\ &\approx \frac{n^2}{2} \left[\left(\frac{2}{n} - \frac{4}{n^2} \right) H_n + 3H_n^{(2)} \right] \sim \frac{\pi^2}{4} n^2. \end{aligned}$$

Finally,

$$Var[T_n] = Var[T_d] + Var[T_n - T_d] \sim \frac{\pi^2}{4} n^2.$$

□

3.3 Experimental results

The expectation of the following SSTs applied to *Random-to-Random Transpositions* shuffling scheme is known:

- Mironov's SST (used in [21]) it is $2nH_n - n$
- `StoppingRuleKLZ` SST it is $n(H_n + 1)$

As shown, both marking schemes are valid for both, *Cyclic-to-Random Transpositions* and *Random-to-Random Transpositions* shufflings. For both stopping rules applied to *Cyclic-to-Random Transpositions* no precise results on expected running times are known. Instead we estimated them via simulations, simply running 10.000 of them. The results are given in Fig. 1.

		`StoppingRuleKLZ`	Mironov's `ST`
ET	$n = 256$	1811	2854
	$n = 512$	3994	6442
	$n = 1024$	8705	14324
$VarT$	$n = 256$	111341	156814
	$n = 512$	438576	597783
	$n = 1024$	1759162	2442503

	$n(H_n + 1)$	$2nH_n - n$
$n = 256$	1823.83	2879.66
$n = 512$	4002.05	6468.11
$n = 1024$	8713.39	14354.79

Fig. 1. Simulations' results for Mironov's and `StoppingRuleKLZ` stopping rules.

The conclusion is that they do not differ much. This suggest following:

Conjecture 1. The mixing time τ_{mix}^{sep} for *Cyclic-to-Random Transpositions* converges to $n \log n$ as $n \to \infty$.

3.4 Optimal shuffling

Cyclic-to-Random Transpositions is the shuffle used in RC4 and in Spritz. To reach stationarity (*i.e.,* produce random permutation), as we have shown, one needs to perform $O(n \log n)$ steps. In each step we use a random number from interval $[0, \ldots, n-1]$, thus this shuffling requires $O(n \log^2 n)$ random bits.

One can ask the following question: Is this shuffling optimal in terms of required bits? The answer is no. The entropy of the uniform distribution on $[n]$ is $O(n \log n)$ (since there are $n!$ permutations of $[n]$), thus one could expect that optimal shuffling would require this number of bits. Applying described *Riffle Shuffle* yields SST with $ET = 2 \lg n$, at each step n random bits are used, thus this shuffling requires $2n \lg n$ random bits, matching the requirement of optimal shuffle (up to a constant).

In Figure 2 we summarized required number of bits for various algorithms and stopping rules.

# bits	RC4	Mironov	KSA$_{KLZ}$	RS
used	$40-2048$			
asympt.		$(2nH_n - 1)\lg n$	$n(H_n+1)\lg n$	$2n \lg n$
required		23 037	14 590	4 096

Fig. 2. Comparison between number of bits used by RC4 (40 to 2048) and required by mathematical models (Mironov [21] and ours – KSA$_{KLZ}$) versus length of the key for the time-reversed riffle shuffle. *Bits asymptotics* approximates the number of fresh bits required by the mathematical model (number of bits required by the underlying Markov chain to converge to stationary distribution). *Bits required* is (rounded) value of *# bits asymptotics* when $n = 256$.

3.5 Perfect shuffle

Let $\mathbf{X}$ be an ergodic chain with a stationary distribution ψ. Ergodicity means that nevertheless of initial distribution, the distribution $\mathcal{L}(X_k)$ converges, in some sense, to ψ as $k \to \infty$ (it is enough to think of $d_{TV}(\mathcal{L}(X_k), \psi)$ converging to 0 as $k \to \infty$). Beside some trivial situations, the distribution of $\mathcal{L}(X_k)$ is *never* equal to ψ for any finite k. That is why we need to study mixing time to know how close $\mathcal{L}(X_k)$ to ψ is. However, there are methods for so-called **perfect simulations**. This term refers to the art of converting a Markov chain into the algorithm which (if stops) returns exact (unbiased) sample from stationary distribution of the chain. The most famous one is the Coupling From The Past (CFTP) algorithm introduced in [27]. The ingenious idea is to realize the chain as evolving *from the past*. However, to apply algorithm effectively the chain must be so-called *realizable monotone*, which is defined w.r.t some underlying partial ordering. Since we consider random walk on permutations, applying CFTP seems to be a challenging task.

However, having Strong Stationary Time rule we can actually *perform perfect simulation*(!). We simply run the chain until event SST occurs. At this time, the Definition 2.2 implies, that the distribution of the chain is stationary.

It is worth mentioning that all SSTs mentioned at the beginning of this Section are deterministic (i.e., they are deterministic functions of the path of the chain, they do not use any extra randomness - it is not the case for general SST). For a method for perfect simulation based on SST (and other methods) see [18], where mainly monotonicity requirements for three perfect sampling algorithms are considered.

4 (NOT SO) RANDOM SHUFFLES OF RC4 – REVISITED

RC4 algorithm. RC4 is a stream cipher, its so-called *internal state* is (S, i, j), where S is a permutation of $[n]$ and i, j are some two indices. As input it takes $L-$byte message $m_1, \ldots, m_L$ and a secret key K and returns ciphertext $c_1, \ldots, c_L$. The initial state is the output of KSA. Based on this state PRGA is used to output bits which are XORed with the message. The actual KSA algorithm used in RC4 is presented in Figure 3 together with its *idealized version* KSA* (where a secret key-based randomness is replaced with pure randomness)

KSA

for $i := 0$ to $n-1$ **do**
  $S[i] := i$
end for

$j := 0$
for $i := 0$ to $n-1$ **do**
  $j := j + S[i] + K[i \bmod l]$

  swap$(S[i], S[j])$
end for
$i, j := 0$

KSA*

for $i := 0$ to $n-1$ **do**
  $S[i] := i$
end for

for $i := 0$ to $n-1$ **do**
  $j :=$**random**(n)
  swap$(S[i], S[j])$
end for
$i, j := 0$

Fig. 3. KSA of RC4 algorithm and its idealized version KSA*.

The goal of KSA of original RC4 is to produce a pseudorandom permutation of $n = 256$ cards. The original algorithm performs 256 steps of *Cyclic-to-Random Transpositions*. Even if we replace $j := j + S[i] + K[i \bmod l]$ in KSA with $j =$ **random**(n) (i.e., we actually consider idealized version), then $\Theta(n \log n)$ steps are needed. Considering idealized version was a main idea of Mironov [21]. Moreover, he showed that $2n \log n$ steps are enough. However, our result stated in Lemma 3 together with experimental results from Section 3.3 imply that around $n \log n$ steps are enough. Moreover, if we consider KSA with *Random-to-Random Transpositions* (instead of *Cyclic-to-Random Transpositions*) then we can state more formal theorem in terms of introduced security definitions.

Theorem 1. Let $\mathcal{A}$ be an adversary. Let $K \in \{0,1\}^{rn}$ be a secret key. Let $\mathcal{S}(K)$ be KSA with *Random-to-Random Transpositions* shuffling which runs for

$$r = n(H_n + 1) + \frac{\pi n}{2} \frac{1}{\sqrt{n! \varepsilon}}$$

steps with $0 < \varepsilon < \frac{1}{n!}$. Then

$$\left| \Pr_{K \leftarrow \{0,1\}^{rn}} [\mathcal{A}(\mathsf{S}(K)) = 1] - \Pr_{R \leftarrow \mathcal{U}(\mathcal{S}_n)} [\mathcal{A}(R) = 1] \right| \leq \varepsilon.$$

Proof. The stopping rule `StoppingRuleKLZ` given in Section 3.2 is an SST for *Random-to-Random Transpositions*, denote it by T. From Lemma 3 we know that $ET = n(H_n + 1) + O(H_n)$ and $VarT \sim \frac{\pi^2}{4} n^2$. Inequality (1) and Chebyshev's inequality imply that for $r = n(H_n + 1) + \frac{\pi n}{2} c$ we have

$$sep(\mathcal{L}(X_r), \psi) \leq P(T > r) \leq \frac{1}{c^2},$$

i.e., $r = \tau^{sep}_{mix}(c^{-2})$. Taking $c = \frac{1}{\sqrt{n!\varepsilon}}$ we perform *Random-to-Random Transpositions* for $r = \tau^{sep}_{mix}(n!\varepsilon)$ steps, *i.e.,* $sep(\mathcal{L}(X_r), \psi) \leq n!\varepsilon$ Inequality (2) implies that $|Pr(X_r = \sigma) - \frac{1}{n!}| \leq \varepsilon$ for any permutation σ and thus completes the proof. □

4.1 Sign distinguisher for or *Cyclic-to-Random Transpositions*.

Sign distinguisher. For a permutation π which has a representation of non-trivial transpositions $\pi = (a_1b_1)(a_2b_2)\dots(a_mb_m)$ the sign is defined as: $sign(\pi) = (-1)^m$. So the value of the sign is $+1$ whenever m is even and is equal to -1 whenever m is odd. The distinguisher is presented in Figure 4.

Fig. 4. Sign distinguisher for *Cyclic-to-Random Transpositions*

Input: S, t
Output: b
if sign(S)=$(-1)^t$
then return *false*
else return *true*

It was observed in [21] that the sign of the permutation at the end of RC4's KSA algorithm is not uniform. And as a conclusion it was noticed that the number of discarded shuffles (by PRGA) must grow at least linearly in n. Below we present this result obtained in a different way than in [21], giving the exact formula for advantage at any step t.

One can look at the sign-change process for the *Cyclic-to-Random Transpositions* as follows: after the table is initialized, sign of the permutation is $+1$ since it is identity so the initial distribution is concentrated in $v_0 = (Pr(\text{sign}(Z_0) = +1), Pr(\text{sign}(Z_0) = -1)) = (1, 0)$.

Then in each step the sign is unchanged if and only if $i = j$ which happens with probability $1/n$. So the transition matrix M_n of a sign-change process induced by the shuffling process is equal to:

$$M_n := \begin{pmatrix} \frac{1}{n} & 1 - \frac{1}{n} \\ 1 - \frac{1}{n} & \frac{1}{n} \end{pmatrix}.$$

This conclusion corresponds to looking at the distribution of the sign-change process after t steps: $v_0 \cdot M_n^t$, where v_0 is the initial distribution. The eigenvalues and eigenvectors of M_n are $(1, \frac{2-n}{n})$ and $(1, 1)^T, (-1, 1)^T$ respectively. The spectral decomposition yields

$$\begin{aligned} v_0 \cdot M_n^t &= (1,0) \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \frac{2-n}{n} \end{pmatrix}^t \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix} \\ &= \left(\frac{1}{2} + \frac{1}{2}\left(\frac{2}{n} - 1\right)^t, \frac{1}{2} - \frac{1}{2}\left(\frac{2}{n} - 1\right)^t\right). \end{aligned}$$

In Fig. 5 the advantage ϵ of a sign-adversary after dropping k bytes of the output (so after $n + k$ steps of the shuffle, for the mathematical model) is presented.

k	$+1$	-1	ϵ
0	.5671382998250798	.4328617001749202	$2^{-3.89672}$
256	.509015	.490985	$2^{-6.79344}$
512	.5012105173235390	.4987894826764610	$2^{-9.69016}$
768	.500163	.499837	$2^{-12.5869}$
1024	.5000218258757580	.4999781741242420	$2^{-15.4836}$
2048	.5000000070953368	.4999999929046632	$2^{-27.0705}$
4096	.5000000000000007	.4999999999999993	$2^{-50.2442}$
8192	.5	.5	$2^{-96.5918}$

Fig. 5. The advantage (ϵ) of the Sign distinguisher of RC4 after $n + k$ steps or equivalently, after discarding initial k bytes.

For $n = 256$ (which corresponds to the value of n used in RC4) and initial distribution being identity permutation after $t = n = 256$ steps one gets: $v_0 \cdot M_{256}^{256} = (0.567138, 0.432862)$.

In [19] it was suggested that the first 512 bytes of the output should be dropped.

5 Strong stationary time based KSA algorithms

In addition to KSA and KSA* (given in Fig. 3) we introduce another version called $\text{KSA}^{**}_{\texttt{Shuffle},\texttt{ST}}(n)$. The main difference is that it does not run a predefined number of steps, but it takes some procedure `ST` as a parameter (as well as `Shuffle` procedure). The idea is that it must be a rule corresponding to Strong Stationary Time for given shuffling scheme. The $\text{KSA}^{**}_{\texttt{Shuffle},\texttt{ST}}(n)$ is given in the following algorithm:

Algorithm 7 $\text{KSA}^{**}_{\texttt{Shuffle},\texttt{ST}}(n)$

Require: Card shuffling `Shuffle` procedure, stopping rule `ST` which is a Strong Stationary Time for `Shuffle`.

for $i := 0$ to $n - 1$ **do**
  $S[i] := i$
end for

while ($\neg$ `ST`) **do**
  `Shuffle`(S)
end while

The algorithm works as follows. It starts with identity permutation. Then at each step it performs some card shuffling procedure `Shuffle`. Instead of running it for a predefined number of steps, it runs until an event defined by a procedure `ST` occurs. The procedure `ST` is designed in such a way that it guarantees that the event is a Strong Stationary Time. At each step the algorithm uses new randomness – one can think about that as of an idealized version but when the length of a key is greater than the number of random bits required by the algorithm then we end up with a permutation which cannot be distinguished from a random (uniform) one (even by a computationally unbounded adversary).

Notational convention: in $\text{KSA}^{**}_{\texttt{Shuffle},\texttt{ST}}(n)$ we omit parameter n. Moreover, if `Shuffle` and `ST` are omitted it means that we use *Cyclic-to-Random Transpositions* as shuffling procedure and stopping rule is clear from the context. Note

that if we use for stopping rule `ST` "stop after n steps" (which of course is *not* SST), it is equivalent to RC4's KSA*.

Given a shuffling procedure `Shuffle` one wants to have a "fast" stopping rule `ST` (perfectly one wants an optimal SST which is stochastically the smallest). The stopping rule `ST` is a parameter, since for a given shuffling scheme one can come up with a better stopping rule(s). This is exactly the case with *Cyclic-to-Random Transpositions* and *Random-to-Random Transpositions*. In Section 3 we recalled Mironov's [21] stopping rule as well as new introduced "faster" rule called (called `StoppingRuleKLZ`).

Let us point out few important properties of $\mathsf{KSA}^{**}_{\texttt{Shuffle},\texttt{ST}}(n)$ algorithm:

- **P1** The resulting permutation is purely random.
- **P2** The information about the number of rounds that an SST-algorithm performs does not reveal any information about the resulting permutation.

The property **P1** is due to the fact that we actually perform *perfect simulation* (see Section 3.5). The property **P2** follows from the fact that the Definition 5.1 is equivalent to the following one (see [4]):

Definition 5.1. Random variable T is **Strong Stationary Time (SST)** if it is a randomized stopping time for chain $\mathbf{X}$ such that:

$$X_T \text{ has distribution } \psi \text{ and is independent of } T.$$

The property **P1** is the desired property of the output of the KSA algorithm, whereas **P2** turns out to be crucial for our timing attacks applications (details in Section 6).

6 Timing attacks and SST

In the black box model (for CPA-security experiment), an adversary has access to an oracle computing encryptions $F(k,\cdot)$ for a secret key k. So the adversary is allowed to obtain a vector of cryptograms $(c_1,\ldots,c_q)$ for plaintexts $(m_1,\ldots,m_q)$ of its choice (where each $c_i = \langle r_i, y_i\rangle = F(k,m_i,r_i)$). Each r_i is the randomness used by the oracle to compute encryption (*e.g.,* its initial counter *ctr* for CTR-mode, *IV* for CBC-mode *etc.*). So the adversary has a vector of tuples

$$\mathcal{O}_{\text{BlackBox},F} = \langle m_i, c_i\rangle = \langle m_i, F(k,m_i,r_i)\rangle\,.$$

In the real world, as a result of interacting with an implementation of F, a timing information may also be accessible to an adversary. In such a case her knowledge is a vector of tuples

$$\mathcal{O}_{Timing,F} = \langle m_i, c_i, t_i\rangle = \langle m_i, F(k,m_i,r_i), t_i\rangle\,,$$

where t_i is the time to reply with c_i on query m_i and randomness used r_i. (A very similar information is accessible to an adversary in chosen-ciphertext attacks.)

An encryption scheme is susceptible to a timing attack when there exists an efficient function H which on input $H\left(\langle m_1,c_1,t_1\rangle,\ldots,\langle m_w,c_w,t_w\rangle\right)$ outputs $\lambda(k)$ (some function of a secret key k). Depending on the scheme and on a particular implementation the leakage function can be of various forms *e.g.,* $\lambda(k) = HW(k)$ – the Hamming weight of k, or in the best case $\lambda(k) = k$.

The reason why a timing attack on implementation of a function F ($y = F(k,x)$) is possible is because the running time of F is not the same on each input. Usually timing attacks focus on a particular part of F – f_{weak} which operates on a function h_x of the original input x and on a function h_k of the original key (*e.g.,* h_k may be a substring of a round key). When $F(k,x) = f_{rest}(k, f_{weak}(k,x))$ the statistical model is built and $T(F(k,x)) = T(f_{weak}(k,x)) + T(f_{rest}(k, f_{weak}(k,x)))$. The distribution of $T(f_{weak}(k,x))$ is conditioned on k and estimating this value (conditioned on subset of bits of k) is the goal of the adversary.

6.1 Countermeasures – approach I

There is one common approach to eliminate possibility of timing attacks, namely to re-implement F ($ct(F) = F_{ct}$ – constant-time implementation of F) in such a way that for all pairs $(k,x) \neq (k',x')$ the running times are the same $T(F(k,x)) = T(F(k',x')) = t$. Then, an adversary's observation:

$$\mathcal{O}_{Timing,ct(F)} = \langle m_i, c_i, t\rangle = \langle m_i, F_{ct}(k,m_i,r_i), t\rangle$$

becomes the same as in the black-box model.

6.2 Countermeasures – approach II

The other approach is to still use a weak implementation, but "blind" (mask) the adversary's view. Let us start with a public-key example, described in the Section 1.1.1. The RSA's fragile decryption $F(d,c) = f_{weak}(d,c)$ (Algorithm 1) is implemented with masking (Algorithm 8) as: $F_{masked}(d,c) = g_2(f_{weak}(d, g_1(c,r)), r)$, where $g_1(c,r) = cr^e \bmod N$ and $g_2(x,r) = xr^{-1} \bmod N$. In this case, the adversary's view is:

$$\mathcal{O}_{Timing,masked(F)} = \langle c_i, F_{masked}(d,c_i), t_i\rangle\,.$$

While t_i are again different, they depend on both d and $g_1(c_i,r_i)$, but an adversary does not obtain information about random r_i's and thous information about t_i is useless.

6.3 Non-constant time implementation immune to timing attacks for symmetric encryption

Let us use the similar approach for the symmetric encryption case. For a time-attack susceptible function F we would like to build a function G by designing functions $g_1(\cdot)$ and $g_2(\cdot)$ and then to define

$$G(k,x) = g_2(F(k, g_1(x,\cdot)),\cdot).$$

We restrict ourselves only to functions G of that form. We do not require that $G(k,x) = F(k,x)$ as it was in the public key case. It would be enough for us that there exists an efficient algorithm for decryption – this is satisfied if both g_1, g_2 have efficiently computable inversions.

Now let us consider the following approaches for selecting g_1 (and g_2):

1) $g_1(x)$ – then such an approach is trivially broken because an attacker knows exactly what is the input to F.

2) $g_1(x, r)$ for a randomly selected r generated during run of G. Assuming that r cannot be efficiently removed from $F(k, g_1(x, r))$ via application of g_2, the ciphertext of G would need to include r but then this is exactly the same case as (1).
3) $g_1(k, x)$ – in that case an attacker might want to perform a timing attack on g_1 – one would need to guarantee that g_1 itself is immune to timing attacks. This approach is used in McBits [7] system, g_1 is a constant-time implementation of Beneš permutation network [6].
4) $g_1(k, x, r)$ – again value r would need to be included in ciphertext generated by G but now for an attacker this case would be no easier than when g_1 depends solely on k (just as in the case 3).

While g_2 cannot remove randomness (unless F is trivially broken) and the randomness used in g_1 needs to be included in the ciphertext, one needs g_2 anyway. It is required for the protection in the case when attacker has access to the decryption oracle – then g_2 plays the same role as g_1 plays for protecting against timing attacks during encryption.

We suggest the following approach, let $k = (k_f, k_g)$ where k_f is a portion of the secret key used by F:

1) Let $g_1(k_g, x), g_2(k_g, x)$ be permutations of $n = |x|$ bits depending on a key. *I.e.,* $g_1(k_g, \cdot)$ and $g_2(k_f, \cdot)$ are permutations constructed based on k_g and k_f respectively, which are later on applied to x, a sequence of n bits (similarly as in McBits scheme)
2) Both g_1, g_2 are the permutations obtained as a result of $\mathsf{KSA}^{**}_{\mathtt{Shuffle,ST}}$ algorithm – one has a freedom in the choice of a shuffle algorithm.

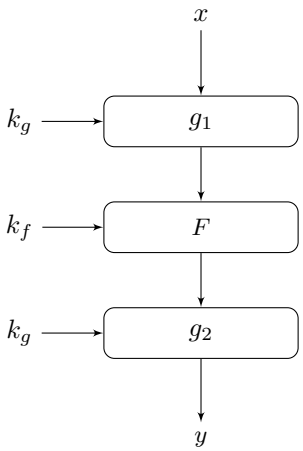

Fig. 6. Timing attack immune G obtained from timing attack susceptible F via compounding with SST-based shuffling algorithms g_1, g_2.

Algorithm 8 G – timing attack immune encryption based on timing attack susceptible symmetric encryption F

Require: x, F, g_1, g_2, k_f, k_g
1: compute $y_1 = g_1(k_g, x)$
2: compute $y_2 = F(k_f, y_1)$
3: **return** $y = g_2(k_g, y_2)$

Algorithm 9 G^{-1} – timing attack immune decryption implementation based on timing attack susceptible symmetric decryption F^{-1}

Require: $y, F^{-1}, g_1^{-1}, g_2^{-1}, k_f, k_g$
1: compute $x_1 = g_2^{-1}(k_g, y)$
2: compute $x_2 = F^{-1}(k_f, x_1)$
3: **return** $x = g_1^{-1}(k_g, x_2)$

Resulting G has the following properties:

G1 Resulting permutation g_1 is uniformly random, provided a long enough k_g is applied (see property **P1** from Section 5).
G2 For a given k_g and for any x, x' one obtains $T(g_1(k_g, x)) = T(g_1(k_g, x'))$ (and the same holds for g_2) – while g_1 is a result of SST, running it for the same input k_g leads to the same running time.
G3 The running time of g_1 (and of g_2) does not reveal any information about the resulting permutation (see property **P2**). As a result, the adversary does not have influence on inputs to F.

The property **G3** guarantees security of the outputs of g_1 and g_2 and that they cannot be derived from the running time but it does no guarantee that k_g will remain secure. It turns out that some bits of k_g may leak through the information about the running time but:

- there are at least $n!$ (where n is the number of bits of x) different keys which have the same running time,
- the information which leaks from the running time does not let to obtain ANY information about the resulting permutation.

For more details see Section 6.5.

6.4 Implementation

To implement the above construction, one needs (pseudo)random bits to generate a permutation g_1. These bits can be obtained in the similar way, one may produce a key for authenticated encryption.

Let k be a 128-bit AES key. Calculate $k_g \leftarrow \text{AES}(k, 0)$, $k_g \leftarrow \text{AES}(k, 1)$, and $k_m \leftarrow \text{AES}(k, 2)$. Then $\text{AES}(k_g, IV)$ is used as a stream cipher to feed *Riffle Shuffle* until SST occurs, producing g_1. The following bits in the same way produce g_2. F is *e.g.,*AES-128 in CBC or CTR mode, all having k_f as a secret key. Then, k_m may be used to compute the MAC (this disallows an adversary to play with *e.g.,* hamming weight of inputs of g_1).

A proof of concept implementation was prepared (as a student project: https://bitbucket.org/dmirecki/bearssl/). This implementation first uses the AES key and riffle-shuffle to generate an array (permutation) which stores the permutation g_1 (of size 128). Then it permutes bits of the buffer by performing 128-step loop (line 34 of the listing 1).

Fix $n = 128 = 2^7$. Any permutation of $[n]$ can be represented as 7 steps of *Riffle Shuffle*. This can be exploited to speed up the application of the permutation:

1) generate a permutation using *Riffle Shuffle* with SST (14 steps on average), say permutation σ was obtained.

Listing 1 A fragment of the (src/riffle.c) responsible for applying a shuffle.

```
/src/riffle.c
34 for (uint32_t i = 0; i < 128; i++) {
35    if (CHECK_BIT(buf[i >> 3], i & 7)) {
36              SET_BIT(temp_buf[permutation[i] >> 3],
37                permutation[i] & 7);
38    }
39 }
```

2) represent σ as 7 steps of some *other Riffle Shuffle* scheme, *i.e.,* compute the corresponding $7 \cdot 128$ bits. Do the same for σ^{-1}.
3) apply this 7-step *Riffle Shuffle* on input bits.

Then the last step may be implemented using efficient instructions: `_pext_u64, _pdep_u64, _mm_popcnt_u64` or its 32-bit analogs. Consult [1] p. 4-270 – 4-273 Vol. 2B for above instructions.

	big	ct	student	sst
AES-128 CBC encrypt	170.78	29.06	11.46	37.63
AES-128 CBC decrypt	185.23	44.28	11.48	37.65
AES-128 CTR	180.94	55.60	16.83	37.52

Fig. 7. Comparison of the efficiency of selected algorithms: `big` – classic, table-based AES implementation, `ct` – constant time implementation using bitslicing strategy, `student` – implementation of the sst-masking described in Section 6 with slow permutation application, `sst` – estimated implementation which of the sst-masking which uses BMI2 and SSE4.2 instructions. All results are in MB/s and were obtained by running speed tests of the BearSSL [25] (via the command: `testspeed all` on Intel i7-4712HQ with 16GB RAM).

6.5 Timing attacks and KSA**

This section explains why bits of the secret key k_g used to generate g_1, g_2 need to be distinct to the bits k_f which are used by F. This is a simple conclusion of properties **P1** and **P2** (defined in Section 5). The examples are intended to give a reader a better insights on the properties achieved by SST shuffling algorithms. As already mentioned - regardless the duration of SST algorithm adversary can learn something about the key, but has completely no knowledge about the resulting permutation. The examples will also show how a choice of a shuffling algorithm influences the leaking information about the key. In both cases we consider extreme (and very unlikely) situation where algorithm runs for the smallest number of steps.

6.5.1 Top-to-random card shuffling

Consider the algorithm $\mathsf{KSA}^{**}_{\mathtt{T2R},\mathtt{ST}}$ with the shuffling procedure corresponding to *Top-To-Random* card shuffling: put the card which is currently on the top to the position j defined by the bits of the key k_g in the current round. Recall stopping rule `ST` which is SST for this shuffle: stop one step after the card which was initially at the bottom reaches the top of the deck. Before the start of the algorithm, mark the last card (*i.e.,* the card n is marked[1]).

In Figure 8 a sample execution of the algorithm is presented (for $n = 8$). It is an extreme situation in which SST was achieved after n steps (the lower bound for the running time) – please keep in mind that the expected running time of this SST is $O(n \log n)$. Based on this knowledge it is easy to observe that there is only one possible value of $k_g[1]$ provided that SST lasted n steps – $k_g[1] = 111 = [8] \setminus [7]$; $k_g[2]$ is either 110 or 111 so $k_g[2] \in [8] \setminus [6]$ and $k_g[i] \in [n] \setminus [n-i]$ for each $i = 1, \ldots, n$. While some information about the key k_g leaks from the fact that the SST stopped just after n steps, still no adversary can learn anything about the resulting permutation because every permutation is generated with exactly the same probability.

We showed an extreme case when SST algorithm for *Top-To-Random* runs the minimal number of steps n. Since $k_g[i] \in [n] \setminus [n-i]$ for $i = 1, \ldots, n$ so the number of bits of entropy of the secret key k_g adversary learns each round i is equal to $\lg i$. There are $n!$ possible keys which result in running time n of SST. An adversary learns a fraction of $n!$ out of $2^{n \lg n} = n^n$ possible keys. Moreover it can predict a prefix of the k_g with high probability.

6.5.2 Riffle Shuffle

Consider the algorithm $\mathsf{KSA}^{**}_{\mathtt{RS},\mathtt{ST}}$ with the shuffling procedure corresponding to (time reversal of) *Riffle Shuffle*. Recall the shuffling: at each step assign each card a random bit, then put all cards with bit assigned 0 to the top keeping the relative ordering. Sample execution is presented in Fig. 9. Recall the construction of SST for this example:

> Consider, initially *unmarked*, all $\binom{n}{2}$ pairs of cards; at each step *mark* pair (i, j) if cards i and j were assigned different bits.

Assume for simplicity that n is a power of 2. How many *pairs* can be marked in first step? For example, if there is only one bit equal to 0 and all other are equal to 1, then $n - 1$ pairs are marked. It is easy to see that the maximal number of pairs will be marked if exactly half (*i.e.,* $\frac{n}{2}$) cards were assign bit 1 and other half were assign bit 0. Then $\frac{n^2}{4}$ pairs will be marked (all pairs of form (i, j): $i < j$ and cards i and j were assigned different bits). After such a step there will be two groups of cards: the ones assigned 0 and ones assigned 1. All pairs of cards within each group are not marked. Then again, to mark as many pairs as possible exactly half of the cards in each group must be assigned bit 0 and other half bit 1 – then $\frac{n^2}{8}$ new pairs will be marked. Iterating this procedure, we see that the shortest execution takes $\lg n$ steps after which all

$$\frac{n^2}{4} + \frac{n^2}{8} + \frac{n^2}{16} + \ldots + \frac{n^2}{2^{1+\lg n}} = \binom{n}{2}$$

pairs are marked. In Figure 9 such sample execution is presented. There are exactly $n!$ keys resulting in $\lg n$ steps of SST (each resulting in different permutation). Thus the adversary learns a tiny fraction of possible keys: $n!$ out of $2^{n \lg n} = n^n$.

6.5.3 Summary

As we have seen the choice of shuffle algorithm matters for KSA^{**}. In any case the resulting permutation is not revealed at all, but some knowledge on key can be obtained. In both presented examples an adversary who observed that the time to generate a permutation was minimal (for a given

1. It is known [3] that optimal SST for *top to random* initially marks card which is second from the bottom.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2017.2751475, IEEE Transactions on Dependable and Secure Computing

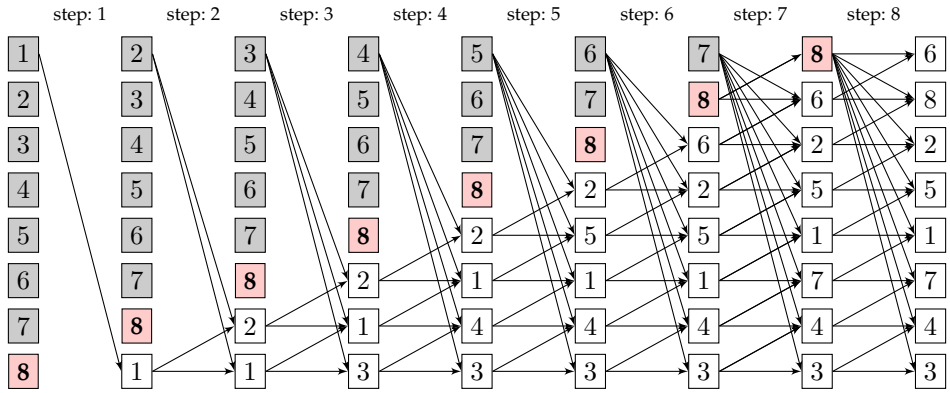

Fig. 8. Example run of *top-to-random* shuffle taken from the key k_g of the length equal to 8 8-value bytes (24-bits). Conditioning on the number of steps of the SST (in this case 8) one can find out that: out of the possible 2^{24} keys only (exactly) $8!$ keys are possible (due to the fact that SST stopped exactly after 8 steps) and every of $8!$ permutations is possible – moreover each permutation with exactly the same probability. SST leaks bits of the key *i.e.,* $k_g[1] = 111$ but does not leak any information about the produced permutation.

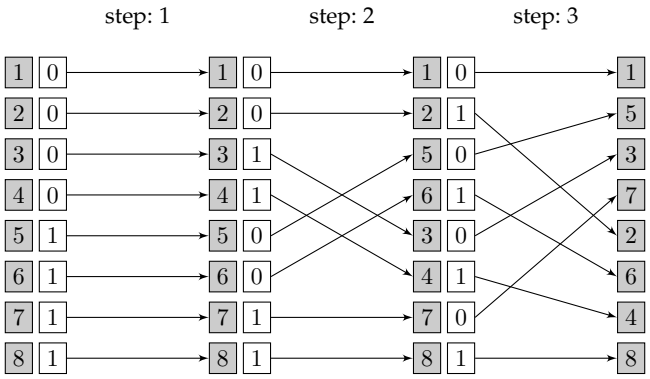

Fig. 9. Example run of (time reversal of) *Riffle Shuffle* of $n = 8$ cards and key $k_g = [00001111, 00110011, 01010101, \ldots]$. At each step the left column represents current permutation, whereas the right one currently assigned bits. See description in the text.

SST) learns that one out of $n!$ keys was used (out of n^n possible keys). However there is a huge difference between these two shuffling schemes:

- in *Riffle Shuffle* an adversary learns that in the first step one key out of $\binom{n}{n/2}$ possible keys (leading to the shortest running time) was used. For the second step there are exactly $\binom{n/2}{n/4}^2$ for each choice of the key made in the fist step, and similarly for the next steps. This makes recovering any information about the key used infeasible.
- in *Top-To-Random* the situation of an adversary is quite different. It gets all $\lg n$ bits of the key used in the first step of the algorithm (provided the algorithm run for n steps). In the ith round it gets information about $\lg n - i + 1$ bits.

Recall that these were the extreme and very unlikely cases. In both cases the probability of the shortest run is equal to $n!/n^n$ ($= 7.29 \times 10^{-55}$ for $n = 128$).

Another advantage of *Riffle Shuffle* is that on average the number of required bits is equal to $2n \lg n$ which is only 2 times more than the minimal number of bits (as presented in the example). For *Top-To-Random* on average the number of bits required is equal to $n \log n \lg n$ which is $\log n$ times larger than the minimal number of bits. This follows from the fact that the variance of SST for *Riffle Shuffle* is much smaller than the variance for *Top-To-Random*. Smaller variance means that fewer information can be gained from the running time. While the expected number of bits required by KSA** may seem to be high, for the post-quantum algorithms like McBits it is not an issue – McBits requires keys of size $2^{16} - 2^{20}$ bytes, *Top-To-Random* needs around 12,000 bits and *Riffle Shuffle* needs on average 4096 bits to generate a uniform permutation for $n = 256$.

It is also worth mentioning that in real-life systems we could artificially additionally mask the duration of the algorithm: If we know values of ET for given SST we can run it always for at least ET steps (similarly, to avoid extremely long execution we can let at most $ET + c \cdot VarT$ steps provided $VarT$ is known).

7 Conclusions

We presented the benefits of using Strong Stationary Times in cryptographic schemes (pseudo random permutation generators). These algorithms have a "health-check" built-in and guarantee the best possible properties (when it comes to the quality of randomness of the resulting permutation).

We showed how SST algorithms may be used to construct timing attack immune implementations out of timing

attack susceptible ones. The SST-based masking leads to comparable (to constant-time implementation) slow-down (but it is platform independent), see Figure 7 for efficiency comparison.

We showed that algorithms using SST achieve better security guarantees than any algorithm which runs predefined number of steps.

We also proved a better bound for the mixing-time of the *Cyclic-to-Random Transpositions* shuffling process which is used in RC4 and showed that different, more efficient shuffling methods (*i.e.*, time reversal of *Riffle Shuffle*) may be used as KSA. This last observation shows that the gap between mathematical model (4096 bits required) and reality (2048 allowed as maximum length of RC4) is not that big as previously thought (bound of 23 037 by Mironov [21]).

REFERENCES

[1] Intel 64 and IA-32 Architectures. Software Developer's Manual, 2016.

[2] Martin R. Albrecht and Kenneth G. Paterson. Lucky Microseconds: A Timing Attack on Amazon's s2n Implementation of TLS. *Advances in Cryptology - EUROCRYPT*, 9665:622—-643, 2016.

[3] David Aldous and Persi Diaconis. Shuffling cards and stopping times. *American Mathematical Monthly*, 93(5):333–348, 1986.

[4] David Aldous and Persi Diaconis. Strong Uniform Times and Finite Random Walks. *Advances in Applied Mathematics*, 97:69–97, 1987.

[5] Nadhem AlFardan, Daniel J Bernstein, Kenneth G Paterson, Bertram Poettering, and Jacob C N Schuldt. On the Security of RC4 in TLS. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 305–320, Washington, D.C., 2013. USENIX.

[6] V.E. Beneš. *Mathematical theory of connecting networks and telephone traffic*. Academic Press, New York :, 1965.

[7] Daniel J Bernstein, Tung Chou, and Peter Schwabe. McBits: fast constant-time code-based cryptography. In *Cryptographic Hardware and Embedded Systems - CHES 2013*, 2013.

[8] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[9] David Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology*, pages 199–203. Springer US, Boston, MA, 1983.

[10] Jean-François Dhem, François Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A Practical Implementation of the Timing Attack. *Smart Card. Research and Applications Third International Conference, CARDIS98*, 1820:167–182, 1998.

[11] Persi Diaconis and Mehrdad Shahshahani. Generating a random permutation with random transpositions. *Zeitschrift fur Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 57(2):159–179, 1981.

[12] James Allen Fill. An interruptible algorithm for perfect sampling via Markov chains. *The Annals of Applied Probability*, 8(1):131–162, February 1998.

[13] S Fluhrer, I Mantin, and A Shamir. Weaknesses in the key scheduling algorithm of RC4. *Selected areas in cryptography*, 2001.

[14] Scott R. Fluhrer and David a. McGrew. Statistical Analysis of the Alleged RC4 Keystream Generator. *Fast Software Encryption, 7th International Workshop*, pages 19–30, 2000.

[15] J. Golic. Linear Statistical Weakness of Alleged RC4 Keystream Generator. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, July 1997.

[16] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. pages 104–113. Springer, Berlin, Heidelberg, 1996.

[17] Michal Kulis, Paweł Lorek, and Filip Zagórski. Randomized stopping times and provably secure pseudorandom permutation generators. In *Mycrypt 2016: Paradigm-shifting Crypto*, pages 145—-167, 2016.

[18] Paweł Lorek and Piotr Markowski. Monotonicity requirements for efficient exact sampling with Markov chains. *To appear in Markov Processes And Related Fields*, 2017.

[19] Itsik Mantin and Adi Shamir. A Practical Attack on Broadcast RC4. *Fast Software Encryption, 8th International Workshop, Yokohama, Japan*, pages 152–164, 2001.

[20] Peter Matthews. A strong uniform time for random transpositions. *J. Theoret. Probab.*, 1(4):411–423, 1988.

[21] Ilya Mironov. (Not So) Random Shuffles of RC4. *Advances in Cryptology—CRYPTO 2002*, 2002.

[22] Elchanan Mossel, Yuval Peres, and Alistair Sinclair. Shuffling by semi-random transpositions. *Foundations of Computer Science*, pages 572–581, 2004.

[23] Cesar Pereida, García Billy, and Bob Brumley. Constant-Time Callees with Variable-Time Callers. Technical report.

[24] Cesar Pereida García, Billy Bob Brumley, and Yuval Yarom. Make Sure DSA Signing Exponentiations Really are Constant-Time. pages 1639–1650, 2016.

[25] Thomas Pornin. BearSSL - Constant-Time Crypto.

[26] Pornin Thomas. BearSSL - Constant-Time Mul.

[27] James Gary Propp and David Bruce Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9:223–252, 1996.

[28] Jacob C. N. Schuldt; Ronald L. Rivest. Spritz—a spongy RC4-like stream cipher and hash function, 2014.

[29] Bart Preneel Souradyuti Paul. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[30] Goutam Paul Subhamoy Maitra. Analysis of RC4 and Proposal of Additional Layers for Better Security Margin. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *Progress in Cryptology - INDOCRYPT 2008*, volume 5365 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[31] Yuval Yarom, Daniel Genkin, and Nadia Heninger. CacheBleed : A Timing Attack on OpenSSL Constant Time RSA. *CHES*, 2016.

[32] Zhijie Shi and R.B. Lee. Bit permutation instructions for accelerating software cryptography. In *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 138–148. IEEE Comput. Soc, 2000.

[33] Bartosz Zoltak. VMPC One-Way Function and Stream Cipher. In *Fast Software Encryption*, pages 210–225. 2004.

Paweł Lorek received PhD degree in applied mathematics from Wrocław University in 2007. From 2008 to 2010 he was a Post-Doctoral fellow at the Department of Mathematics and Statistics, University of Ottawa. Currently he is an Assistant Professor at the Mathematical Institute of the University of Wrocław. His main research interests focus on Markov chains (speed of convergence to stationarity, stability of the system, asymptotics of stationary distribution), their applications in cryptography, randomized algorithms, simulations and image analysis.

Filip Zagórski received PhD degree in mathematics (in the area of computer science) in 2010. From 2010 to 2011 he was a Post-Doctoral fellow at the Department of Computer Science of The George Washington University. Currently he is an Assistant Professor at the Computer Science Department of the Wroclaw University of Science and Technology. His main research interests focus on cryptography *i.e.*, application of Markov chains to cryptographic schemes and on security *i.e.*, constructions of voter verifiable voting schemes.

Michał Kulis a bachelor of Computer Science and a student of Mathematical Statistics at Wroclaw University of Technology. His main research interests focus on probability theory, algorithms and Markov chains. Currently he is working on his master's diploma regarding estimation of distances between probability distributions.