Provable randomness over lightweight permutations

Juan Di Mauro, Hamid Boukkerou, Gilles Millerioux, Marine Minier, Thomas Stoll

Provable randomness over lightweight permutations

Juan Di Mauro[3], Hamid Boukkerou[1], Gilles Millerioux[1], Marine Minier[2*], Thomas Stoll[3]

[1]Université de Lorraine, CNRS, CRAN, Nancy, France.
[2]Université de Lorraine, CNRS, Inria, LORIA, Nancy, France.
[3]Université de Lorraine, CNRS, Inria, IECL, Nancy, France.

*Corresponding author(s). E-mail(s): marine.minier@loria.fr;

Abstract

Permutations mostly in the form of sboxes are very efficient components largely present in cryptographic artifacts either hardware or software. They have been well studied as an individual component and have a large body of knowledge. Here we look at the properties of tiny transformations employing permutations and a few field operations. That gives functions with few components, which can be safely used either as PRG building blocks or as a component in lightweight ciphers. At the end, we propose a PRG implementation with those components. Lastly, we formalize a key assumption over the system functions and give a theorem for the equivalence of some system instances.

Keywords: randomness, provable, permutations

1 Introduction

Randomness is a key component of any relevant cryptographic device. True randomness is very expensive and usually out of reach, so in most cases, an algorithm is used to derive elements that look random. These procedures usually take a starting value which is required to be not biased and they produce a sequence of samples that looks like drawing from a uniform distribution.

Computational indistinguishability of two distributions X, Y is the negligible success of any probabilistic polynomial time (ppt) algorithm to distinguish between both (given only samples) and is the security model for randomness devices (typically the

samples for X are the output of the procedure and Y is the uniform distribution). Intuitively, if the algorithm is *good*, it will be hard to tell the difference between a set of truly random samples and the output of the procedure.

Indistinguishability in its formal definition implies the non-existence of any ppt distinguisher. Such a property can only be proven by formal means and when that is hard to achieve, the impossibility of a large set of distinguishers (such as [1], [2]) to tell apart the output of a (pseudo)random generator from the uniform distribution is taken as evidence that supports the hypothesis that the sequence produced is indistinguishable from a random distribution.

Recently, the use of machine learning techniques seems to improve at least in a practical way the construction of distinguishers [3],[4]. So, from the point of view of a cryptoanalyst, is becoming more feasible to find good distinguishers and enlarge the set of tools for the analysis.

That means that, unless a completeness proof of some statistical test can be given, the support evidence given by a set of statistical tests, no matter how large, is becoming a weaker argument, so a formal proof of the no-existence of distinguishers were and are very important in cryptographic design.

But, good random quality is not only important for random generators. From a theoretical point of view, several equivalences between primitives (see Sections 4.5 and 4.6 in [5]) are the main argument for relating biases with weakness. From a practical side, a bias can be exploited to build a distinguisher [6] and distinguishers are frequently used to build active and harmful attacks over ciphers [7].

Despite the design of random generators with good statistical properties is far from trivial, they need to be fast as well. That is quite clear because the final destination is to be used as a part of a real-life system. In recent years, that is more important given the need to incorporate security into devices with a modest computing power (such as the ones found in IoT). The area of lightweight cryptography is devoted to finding designs that meet security standards with very low complexity and energy consumption. Constructions that use few, affordable, and fast operations are increasing in importance.

This work was motivated by the self-synchronizing stream cipher (for short, SSSC) presented in [8]. There, the authors present an SSSC that consists mainly of a dynamical system that starts from an initial state and evolves with a keyed function transition (so that the next state is a keyed transition from the actual state and some public but time-dependent parameter).

Good distribution in the sequence of states of the system is a requirement. However, it is unclear from the previous work if that is true and under which conditions.

The state transition proposed there can be generalized become important on its own. Our first contribution here is to show that the general form of that equation preserves the randomness of its arguments. Since they are very modest in terms of the number of operations and complexity, that gives a suitable generic component that can be reused in other systems.

Secondly, we instantiate a pseudo-random generator using those elements and give formal proof of its security. This accomplishes the former missing goal in [8] showing that the state transitions are sufficiently random.

Lastly, we revisit the theory of some PRG based on one-way functions to give bounds for the output of our generator. The clear analogy between the last topic and the maximality results given in [9] is remarkable since results from the latter are usually theoretical.

The paper is organized as follows: Section 2 recalls the security notions relevant to our analysis and additional preliminaries. In section 3 we present the general update equation and state its random-preservation property. Section 4 presents a generator with those components and shows its security. Finally, section 5 deals with the output bounds and the relations with one-wayness assumptions.

2 Preliminaries and models of security

We recall here some usual notation for security proofs.

Probabilistic polynomial time (ppt) adversaries:

A probabilistic polynomial time (ppt) *adversary* $\mathcal{A}$ is a ppt interactive Turing machine.

We will model the interaction between adversary $\mathcal{A}$ and the system with a sequence of steps. In the end, $\mathcal{A}$ will output a bit that is the result of its computations, with the given information provided by the system. The sequence of steps prior to giving the information to $\mathcal{A}$ is written as $S_1; S_2; \ldots$. So,

$$Pr[S_1; S_2; \ldots; S_n; \mathcal{A}(\bar{x}) = 1]$$

is the probability that $\mathcal{A}$ outputs 1 after the steps $S_1, \ldots, S_n$ with the input $\bar{x}$ that (usually) depends on the previous n steps.

Distinguishers advantage:

For a random variable X, we write $x \leftarrow X$ to say that x is a value taken from X according to its distribution. We also write $x \xleftarrow{r} R$ to say that x is extracted from some set R according to the uniform distribution.

Let $\mathcal{A}$ be a ppt adversary. The advantage of $\mathcal{A}$ in distinguishing two random distributions X, Y is given by

$$\mathrm{Dist}\mathtt{adv}(\mathcal{A}, X, Y) = |Pr[x \leftarrow X; \mathcal{A}(x) = 1] - Pr[y \leftarrow Y; \mathcal{A}(y) = 1]|$$

We will omit Y if Y is a uniform random variable with the same range as X and simply write $\mathrm{Dist}\mathtt{adv}(\mathcal{A}, X)$.

Pseudo-random generators advantage:

A PRG can be defined in an abstract way as a mapping $G : \{0,1\}^l \to \{0,1\}^L$ with $l < L$, which can be efficiently computed. The argument of G is called the seed.

For any ppt algorithm $\mathcal{A}$ the advantage of $\mathcal{A}$ over the PRG G is the probability that $\mathcal{A}$ succeeds in distinguishing $G(s)$ from a random sample over $\{0,1\}^L$, provided that s is chosen randomly:

$$\mathrm{PRGadv}(\mathcal{A}, G) = \left| Pr\left[\begin{array}{l} s \xleftarrow{r} \{0,1\}^l; \\ y \leftarrow G(s); \\ \mathcal{A}(y) = 1 \end{array}\right] - Pr\left[\begin{array}{l} y \xleftarrow{r} \{0,1\}^L; \\ \mathcal{A}(y) = 1 \end{array}\right] \right| \quad (1)$$

We will call Experiment 0 the one where the output of the PRG is given to the adversary, and Experiment 1 the one where the output of the PRG is replaced by a random string.

In detail, in Experiment 0 the adversary is given as input a value y computed as follows

1. $s \xleftarrow{r} \{0,1\}^l$
2. $y \leftarrow G(s)$

On the other hand, in Experiment 1 the adversary is given as input a value $y \xleftarrow{r} \{0,1\}^L$. The advantage of $\mathcal{A}$ is, therefore, the difference between the probability of output 1 in Experiment 0 and the probability of output 1 in Experiment 1.

Negligible functions:

A function $f : \mathbb{N} \to \mathbb{R}$ is negligible if for every polynomial p there exists an N such that for all $n > N$ we have $f(n) < \frac{1}{p(n)}$.

We will denote a negligle (unspecified) function as `negl`.

3 General form of the transformations

This section describes our main artifact. The construction is taken mostly from the work in [8]. There the authors present a self-synchronizing stream cipher with a matrix transformation that updates the system and an output function that serves at the same time as feedback for the next iteration. The matrix itself is changed on every iteration using the feedback on the output function.

From a highlevel perspective, the system is composed of an internal state, a matrix transformation and an output function. All the operations are performed in a finite field $\mathbb{F} = GF(2^q)$ (choice of q is not important for our analysis and we leave it as an unspecified parameter for the moment). The internal state is a vector $\mathbf{x}$ of n elements in the field $\mathbb{F}$. The matrix transformation is a $n \times n$ matrix $\mathbf{A}$ with elements in $\mathbb{F}$. The output function is a function $\mathcal{O} : \mathbb{F}^n \to \mathbb{F}^p$ that takes the internal state and produces the output as a vector with $p \leq n$ elements in the field (the dimension p will not be relevant until Section 5). Below we describe the components of the system in more detail.

State transitions:

The transition matrix $\mathbf{A}$ is updated on every iteration and after that, used to obtain a new internal state $\mathbf{x}$. We will denote those transitions using the subscript t to indicate

the iteration, so that $\mathbf{A}_t$ is the matrix transformation at iteration t and the internal state at iteration t is $\mathbf{x}_t$. At iteration t the product $\mathbf{A}_t\mathbf{x}_t$ gives a new state $\mathbf{x}_{t+1} \in \mathbb{F}^n$.

Output function $\mathcal{O}$:

The output function takes the internal state $\mathbf{x}_{t+1}$ after the transition and produces the output $\mathcal{O}(\mathbf{x}_{t+1}) = \mathbf{y}_{t+1} \in \mathbb{F}^p$. To keep the procedure consistent, we will take as first output $\mathbf{y}_0 = \mathcal{O}(\mathbf{x}_0)$.

Parameter function $\mathcal{M}$:

Since part of our analysis will be focused on output dimensions, we will need an additional component that can map the output of $\mathcal{O}$ to a field element. This requirement is not present in the original stream cipher in [8] because the output function is instantiated with $p = 1$, but we need to be more general in our analysis so we can study the relation between the output dimension p and the security of the system. We will denote this additional component as $\mathcal{M} : \mathbb{F}^p \to \mathbb{F}$ and we will take the output of the system after $\mathcal{M}$ as $c_t = \mathcal{M}(\mathbf{y}_t)$.

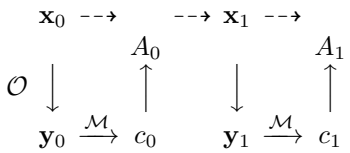
Keyed update function $\mathcal{U}_k$:

The procedure to generate the transition matrix at each step is a key part of the system. The matrix at step t is a linear mapping $\mathbf{A}_t : \mathbb{F}^n \to \mathbb{F}^n$ that depends on the output $\mathbf{y}_t$ and a secret key $k \in \mathbb{F}^l$ (we will leave l undetermined by now). A natural way of formalize this is to define a keyed update function $\mathcal{U}_k : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}^n$ that takes the previous internal state $\mathbf{x}_t$, the output value c_t and outputs a new internal state $\mathbf{x}_{t+1}$. The linear mapping that performs the internal state transition $\mathbf{x}_t \mapsto \mathbf{x}_{t+1}$ at time t, represented by the matrix $\mathbf{A}_t$ can be naturally expressed by $\mathcal{U}_k(\cdot, c_t)$ such that

$$\mathcal{U}_k(\mathbf{x}_t, c_t) = \mathbf{A}_t\mathbf{x}_t.$$

Figure 1 shows a schematic diagram of the state transitions.

Fig. 1 Schematic picture of the iteration procedure. Dotted arrows denote state transition, solid arrows with function symbols shows results of applying the corresponding function. The arrows between c_t and A_t for $t = 0, 1$ denote the effect of c_t in the construction of A_t as indicated by the update function $\mathcal{U}_k(\cdot, c_t)$.



We will turn our attention now to individual components of the state vector $\mathbf{x}_t$ and the way they are updated. Without diving into the details of how the transition matrix is constructed, the component j of the state vector $\mathbf{x}_{t+1}$ is the result of applying some linear function to the previous state $\mathbf{x}_t$. Since our matrix A_t depends on c_t and the key

k, we will add those dependencies to the notation and we will denote as $f_j(\mathbf{x}_t; k, c_t)$ the function that updates the component j of the state vector where f_j is a linear function in the first argument, that depends on the output c_t and the secret key k.

In general, the form of the update function f_j can be written as

$$f_j(\mathbf{x}_t; k, c_t) = \sum_{i=1}^{n} a_{ji}(k, c_t)\mathbf{x}_{t,i}$$

where $a_{ji}(k, c_t)$ are field elements that depend on the key and the output. The element j, i of the matrix $\mathbf{A}_t$ at iteration t is therefore $a_{ji}(k, c_t)$ and $\mathbf{x}_{t,i}$ denotes the component i of the state vector $\mathbf{x}_t$ (at time t).

To introduce non-linearity in the system we will use a function $S : \mathbb{F} \to \mathbb{F}$ to define some of the entries of the matrices $\mathbf{A}_t$. We will require S to be a permutation.

Regarding the key elements, we will assume that k has enough elements to cover all the entries of the matrix $\mathbf{A}_t$. That is, we will take k as an vector in $\mathbb{F}^{n^2}$ and sorted as $k_{11}, k_{12}, \ldots, k_{1n}, k_{21}, \ldots, k_{nn}$. We will like to remark that the size of the key space is larger than we actually need, but we will not use this fact in our analysis. Moreover, specifying the exact number of key bits needed would only introduce cumbersome notation in our proofs.

The matrix elements $a_{ji}(k, c_t)$ are defined as follows: for every j we choose an index $j_0 \in \{1, \ldots, n\}$ and a subset $I_j \subset \{1, \ldots, n\}$ of size $|I_j| = n - 1$ such that $j_0 \notin I_j$.

$$a_{ji}(k, c_t) = \begin{cases} 1 & \text{if } i = j_0 \\ S(c_t + k_{ji}) & \text{if } i \in I_j \\ 0 & \text{otherwise} \end{cases}$$

Where k_{ji} is the ji component of the key k. This leads to the expression

$$f_j(\mathbf{x}_t, k, c_t) = \mathbf{x}_{t,j_0} + \sum_{i \in I_j} \mathbf{x}_{t,i} S(c_t + k_{ji}) \tag{2}$$

A toy example

Let $n = 3$ and $I_1 = \{2, 3\}$, $I_2 = \{1, 3\}$, $I_3 = \{1, 2\}$. Take $j_0 = j$ for $j = 1, 2, 3$. The update function for the first component of the state vector is

$$f_1(\mathbf{x}_t, k, c_t) = \mathbf{x}_{t,1} + \mathbf{x}_{t,2} S(c_t + k_{1,2}) + \mathbf{x}_{t,3} S(c_t + k_{1,3})$$

For the second component we have

$$f_2(\mathbf{x}_t, k, c_t) = \mathbf{x}_{t,2} + \mathbf{x}_{t,1} S(c_t + k_{2,1}) + \mathbf{x}_{t,3} S(c_t + k_{2,3})$$

And for the third component

$$f_3(\mathbf{x}_t, k, c_t) = \mathbf{x}_{t,3} + \mathbf{x}_{t,1} S(c_t + k_{3,1}) + \mathbf{x}_{t,2} S(c_t + k_{3,2})$$

The matrix $\mathbf{A}_t$ at iteration t is given by

$$\mathbf{A}_t = \begin{pmatrix} 1 & S(c_t + k_{1,2}) & S(c_t + k_{1,3}) \\ S(c_t + k_{2,1}) & 1 & S(c_t + k_{2,3}) \\ S(c_t + k_{3,1}) & S(c_t + k_{3,2}) & 1 \end{pmatrix}.$$

A simpler example of an output function is $\mathcal{O} : \mathbb{F}^3 \to \mathbb{F}$ given by the mapping $\mathbf{x}_t \mapsto \mathbf{x}_{t,3}$ and the function $\mathcal{M} : \mathbb{F} \to \mathbb{F}$ given by the identity.

According to this, matrix $\mathbf{A}_t$ can be rewritten as

$$\mathbf{A}_t = \begin{pmatrix} 1 & S(\mathbf{x}_{t,3} + k_{1,2}) & S(\mathbf{x}_{t,3} + k_{1,3}) \\ S(\mathbf{x}_{t,3} + k_{2,1}) & 1 & S(\mathbf{x}_{t,3} + k_{2,3}) \\ S(\mathbf{x}_{t,3} + k_{3,1}) & S(\mathbf{x}_{t,3} + k_{3,2}) & 1 \end{pmatrix}.$$

Lastly, the output of the system at time $t + 1$ is given by

$$\mathcal{O}(\mathcal{U}_k(\mathbf{x}_t, c_t)) = \mathcal{O}(\mathbf{A}_t\mathbf{x}_t) = \mathbf{x}_{t+1,3} = \mathbf{x}_{t,3} + \mathbf{x}_{t,1}S(\mathbf{x}_{t,3} + k_{3,1}) + \mathbf{x}_{t,2}S(\mathbf{x}_{t,3} + k_{3,2}).$$

Functions like the one in Equation (2) are frequently used as tiny components to build larger systems. We remark that the overall complexity of (2) is very affordable. The function S can be an S-box, and this in turn can be implemented as a table either in software or hardware. Multiplications and additions in the extension fields of $\mathbb{F}_2$ are also, very efficient (via logarithmic and Zech logarithmic tables).

Having said that, our main concern is about the randomness of the sequence of outputs $\mathbf{y}_t$ produced at each iteration. First of all, it is not clear if the sequence has good statistical quality. Secondly, assuming that the output function $\mathcal{O}$ and function $\mathcal{M}$ do not play any cryptographic role, it is unclear how large should be the output of the system to remain indistinguishable from a random sequence.

In the original design of the stream cipher, the authors propose to use as output function the mapping $\mathbf{x}_t \mapsto \mathbf{x}_{t,3}$ so, in that case $p = 1$ and the function $\mathcal{M}$ can be the identity. However, each iteration involves several operations, so it will be an improvement if p can be larger than 1 and at the same time, security is preserved.

To address the question about statistical quality, we will prove something stronger. We will prove that if the initial state $\mathbf{x}_0$ is computationally indistinguishable from a random vector in $\mathbb{F}^n$ and the parameters c_t are computationally indistinguishable from a random element in $\mathbb{F}$, then the sequence of states $\mathbf{x}_t$ is computationally indistinguishable from a sequence of random vectors in $\mathbb{F}^n$. In other words, our system preserves the randomness of its arguments.

The following two lemmas and its corollary are devoted to prove that mappings in the form of Equation (2) preserve the randomness of the input state. This is done with increasing complexity, starting with the simplest case and then generalizing to the case of several components.

The simplest form of Equation (2) is the one given by

$$f_j(\mathbf{x}_t, k, c) = \mathbf{x}_{t,j_0} + \mathbf{x}_{t,j_1}S(c_t + k_{j,j_1}). \tag{3}$$

For convenience in the notation, let X_l be the random variable for the component $\mathbf{x}_{t,l}$ of the state in the current iteration t. For example, Equation (3) involves the random variables X_{j_0} and X_{j_1}.

Lemma 1. *Assume that the component function f_j has the form given in Equation 3 and that k_{j,j_1} is chosen uniformly at random from $\mathbb{F}$. If S is a permutation then for every adversary $\mathcal{A}$ that can distinguish the output of f_j there exist two adversaries $\mathcal{B}', \mathcal{B}$ such that*

$$\mathrm{Dist}^{\boldsymbol{adv}}(\mathcal{A}, f_j(\mathbf{x}_t, k, c)) \leq \mathrm{Dist}^{\boldsymbol{adv}}(\mathcal{B}', X_{j_0}) + \mathrm{Dist}^{\boldsymbol{adv}}(\mathcal{B}, X_{j_1}).$$

Proof. The outline of the proof is as follows: we will use a sequence of experiments to build a hybrid argument. In the first experiment, the output of f_j is replaced by $r_0 + r_1 r_2$ for three random values r_0, r_1, r_2. In the next experiments, we replace in turn each one of these with the real values used in the computation of f_j. Any noticeable difference between the experiments leads to a distinguisher for the element that changes.

Let E_0 be the experiment given by the following steps

1. $u \xleftarrow{r} \mathbb{F}$
2. $v \xleftarrow{r} \mathbb{F}$
3. $w \xleftarrow{r} \mathbb{F}$
4. Compute $z \leftarrow u + wv$ and send it to $\mathcal{A}$

The next four experiments are listed below

E_1 :

1. $u \xleftarrow{r} \mathbb{F}$
2. $v \xleftarrow{r} \mathbb{F}$
3. $w \leftarrow S(k_{j,j_1} + c_t)$
4. Compute $z \leftarrow u + wv$ and send it to $\mathcal{A}$

E_2 :

1. $u \xleftarrow{r} \mathbb{F}$
2. $v \leftarrow \mathbf{x}_{t,j_1}$
3. $w \leftarrow S(k_{j,j_1} + c_t)$
4. Compute $z \leftarrow u + wv$ and send it to $\mathcal{A}$

E_3 :

1. $u \leftarrow \mathbf{x}_{t,j_0}$
2. $v \xleftarrow{r} \mathbb{F}$
3. $w \leftarrow S(k_{j,j_1} + c_t)$
4. Compute $z \leftarrow u + wv$ and send it to $\mathcal{A}$

E_4 :

1. $u \leftarrow \mathbf{x}_{t,j_0}$

2. $v \leftarrow \mathbf{x}_{t,j_1}$
3. $w \leftarrow S(k_{j,j_1} + c_t)$
4. Compute $z \leftarrow u + wv$ and send it to $\mathcal{A}$

Let W_b be the event that $\mathcal{A}$ outputs 1 in experiment E_b for $b = 0, 1, 2, 3, 4$. Clearly,

$$\text{Dist}\mathtt{adv}(\mathcal{A}, f_j(\mathbf{x}_t, k, c_t)) = |Pr[W_0] - Pr[W_4]| \tag{4}$$

We can write $|Pr[W_0] - Pr[W_4]|$ as the following sum of differences

$$\begin{aligned} |Pr[W_0] - Pr[W_4]| &= |Pr[W_0] - Pr[W_1] \\ &\quad + Pr[W_1] - Pr[W_2] \\ &\quad + Pr[W_2] - Pr[W_4] \\ &\leq |Pr[W_0] - Pr[W_1]| \\ &\quad + |Pr[W_1] - Pr[W_2]| \\ &\quad + |Pr[W_2] - Pr[W_4]|. \end{aligned} \tag{5}$$

We point out that

- $|Pr[W_0] - Pr[W_1]| = 0$ because u, v are chosen uniformly at random from $\mathbb{F}$.
- $|Pr[W_1] - Pr[W_2]| = 0$ since both experiments are statistically identical because u is chosen uniformly at random.

Then, last inequality in Equation (5) gives us

$$|Pr[W_0] - Pr[W_4]| \leq |Pr[W_2] - Pr[W_4]|. \tag{6}$$

In an almost identical way, we can write $|Pr[W_0] - Pr[W_4]|$ as the following sum of differences

$$\begin{aligned} |Pr[W_0] - Pr[W_4]| &= |Pr[W_0] - Pr[W_1] \\ &\quad + Pr[W_1] - Pr[W_3] \\ &\quad + Pr[W_3] - Pr[W_4] \\ &\leq |Pr[W_0] - Pr[W_1]| \\ &\quad + |Pr[W_1] - Pr[W_3]| \\ &\quad + |Pr[W_3] - Pr[W_4]|. \end{aligned} \tag{7}$$

And as before, we have that $|Pr[W_1] - Pr[W_3]| = 0$ because v is randomly chosen with uniform probability over $\mathbb{F}$. Then, we have

$$|Pr[W_0] - Pr[W_4]| \leq |Pr[W_3] - Pr[W_4]|. \tag{8}$$

Equations (6) and (8) give us that

$$|Pr[W_0] - Pr[W_4]| \leq |Pr[W_2] - Pr[W_4]| + |Pr[W_3] - Pr[W_4]|. \tag{9}$$

To finish the proof, we need to show that $|Pr[W_3] - Pr[W_4]| \leq \text{Dist}\mathtt{adv}(\mathcal{B}, X_{j_1})$ and $|Pr[W_2] - Pr[W_4]| \leq \text{Dist}\mathtt{adv}(\mathcal{B}', X_{j_0})$.

In fact, is straightforward to build from $\mathcal{A}$ an adversary $\mathcal{B}'$ that can distinguish X_{j_0} from the uniform distribution from experiments E_2 and E_4, so $|Pr[W_2] - Pr[W_4]| = \text{Dist}\mathtt{adv}(\mathcal{B}', X_{j_0})$.

Adversary $\mathcal{B}'$ acts in turn as a challenger for $\mathcal{A}$ in the experiment E_2 and E_4, following this steps

1. Receives u from its own challenger.
2. Pick v uniformly at random from $\mathbb{F}$.
3. Compute $w \leftarrow S(k_{j,j_1} + c_t)$.
4. Compute $z \leftarrow u + wv$ and send z to $\mathcal{A}$.
5. Output the same as $\mathcal{A}$.

When u is sampled from the uniform distribution, the output of $\mathcal{B}'$ is the output of $\mathcal{A}$ in E_2. Conversely, when u is sampled from the distribution of X_{j_0}, the output of $\mathcal{B}'$ is the output of $\mathcal{A}$ in E_4. Then, it holds that

$$|Pr[W_2] - Pr[W_4]| = \text{Dist}\mathtt{adv}(\mathcal{B}', X_{j_0}). \tag{10}$$

In a very similar way we can build a procedure $\mathcal{B}$ using $\mathcal{A}$, that can distinguish X_{j_1} from the uniform distribution from experiments E_3 and E_4, such that

$$|Pr[W_3] - Pr[W_4]| = \text{Dist}\mathtt{adv}(\mathcal{B}, X_{j_1}). \tag{11}$$

Finally Lemma follows from inequality in Equation (9) and Equations (4), (10) and (11). □

The argument of the above Lemma can be extended to the general case

$$f_j(\mathbf{x}_t, k, c_t) = \mathbf{x}_{t,j_0} + \sum_{i \in I_j} \mathbf{x}_{t,i} S(c_t + k_{j,i}). \tag{12}$$

Lemma 2. *Assume that the component function f_j has the form given in Equation (12) and that for every $i \in I_j$, $k_{j,i}$ is chosen uniformly at random and independently from $\mathbb{F}$. Let S be a permutation. For each $i \in I_j$, let*

$$\epsilon_i = \sup_{\mathcal{B}} \{\text{Dist}\mathtt{adv}(\mathcal{B}, X_i)\}$$

where the supremum is taken over all the ppt distinguisher adversaries for X_i. Let also,

$$\epsilon = \max\{\epsilon_i \mid i \in I_j\}.$$

Then for every distinguisher adversary $\mathcal{A}$ for the output of f_j we have

$$\mathrm{Dist}\,\mathtt{adv}(\mathcal{A}, f_j(\mathbf{x}_t, k, c_t)) \leq |I_j|\epsilon.$$

Since outputs of functions f_j will be the components of the new state $\mathbf{x}_{(t+1)}$, we will use the same notation regarding random variables and we will call $\hat{X}_j$ the random variable for the output of f_j.

The following Corollary is derived easily from the previous Lemma.

Corollary 1. *With the same notations of Lemma 2, if every component $\mathbf{x}_{t,i}$ with $i \in I_j$ is computationally indistinguishable of sample from the uniform distribution over $\mathbb{F}$, then the distribution for the updated component $\hat{X}_j$ is also computationally indistinguishable from a uniform distribution over the same range.*

Proof of Lemma 2. By induction on $|I_j|$. The case $|I_j| = 1$ is Lemma 1. For $|I_j| > 1$, take any $j_1 \in I_j$ and let us write $I'_j = I_j \setminus \{j_1\}$. Then

$$\mathbf{x}_{t+1,j} = \mathbf{x}_{t,j_0} + \sum_{i \in I_j} \mathbf{x}_{t,i} S(c_t + k_{ji}) = \mathbf{x}_{t,j_0} + \sum_{i \in I'_j} \mathbf{x}_{t,i} S(c_t + k_{ji}) + \mathbf{x}_{t,j_1} S(c_t + k_{j,j_1} + c)$$

Let f'_j be the same as f_j but summing over I'_j instead of I_j, that is

$$f'_j(\mathbf{x}_t; k, c_t) = \mathbf{x}_{t,j_0} + \sum_{i \in I'_j} \mathbf{x}_{t,i} S(c_t + k_{ji}).$$

Let $\mathcal{A}$ be any distinguisher adversary for the output of f_j. Since

$$f_j(\mathbf{x}_t; k, c_t) = f'_j(\mathbf{x}_t; k, c_t) + \mathbf{x}_{t,j_1} S(c_t + k_{j,j_1} + c)$$

one application of Lemma 1 gives us two distinguisher adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that

$$\mathrm{Dist}\mathtt{adv}(\mathcal{A}, f_j(\mathbf{x}_t; k, c_t)) \leq \mathrm{Dist}\mathtt{adv}(\mathcal{B}', f'_j(\mathbf{x}_t; k, c_t)) + \mathrm{Dist}\mathtt{adv}(\mathcal{B}, X_{j_1}) \tag{13}$$

By the induction hypothesis over f'_j, for any distinguisher adversary $\mathcal{A}'$ it follows that

$$\mathrm{Dist}\mathtt{adv}(\mathcal{A}', f'_j(\mathbf{x}_t; k, c_t)) \leq |I'_j|\epsilon' \text{ , for } \epsilon' = \max\{\epsilon_i \mid i \in I'_j\}. \tag{14}$$

That holds for $\mathcal{B}'$ in Equation (**??**) in particular. Then, we have

$$\mathrm{Dist}\mathtt{adv}(\mathcal{A}, f_j(\mathbf{x}_t; k, c_t)) \leq |I'_j|\epsilon' + \mathrm{Dist}\mathtt{adv}(\mathcal{B}, X_{j_1}). \tag{15}$$

Finally, the Lemma follows by the fact that

$$\max(\epsilon', \mathrm{Dist}\mathtt{adv}(\mathcal{B}, X_{j_1})) \leq \epsilon$$

□

Remark about key size

Analysis in this section was done considering a key with same amount of elements as the matrix $\mathbf{A}_t$. That was done to simplify the notation, specially in the proofs. However, it should be noted that such key size is far larger than needed. The set of equations given by Equation **??** with $j = 1, \ldots, n$ needs $|I_1| + |I_2| + \ldots + |I_n|$ key elements. But sets I_j are at most of size $n - 1$ and usually much smaller. Original work in [8] establish a proper key size for the system. In our case, we are mainly interested in the randomness of the system and the relation between the number of outputs components and the key size. From now on, we will take as $\mathbb{F}^l$ the key space.

4 PRG instance

Informally, each instance of Equation (12) gives a PRG that stretches the seed $\mathbf{x}_t$ by many bits as the size of elements in $\mathbb{F}$. Given two parameters $k \in \mathbb{F}^l$ and $c \in \mathbb{F}$, we will name as G_j the PRG given by the mapping $\mathbf{x}_t \mapsto f_j(\mathbf{x}_t, k, c)$. In what follows, we will drop the subscript t that indicates the iteration and we will only keep a subscript for the index of the elements in the vectors.

By Lemma 2, each G_j is a secure PRG (taking $\mathbf{x}$ as the seed element). Let G be the mapping given by $\mathbf{x} \mapsto (G_1(\mathbf{x}), G_2(\mathbf{x}), \ldots, G_n(\mathbf{x}))$. The complete state update

$$\mathbf{x} \mapsto G(\mathbf{x}) = (G_1(\mathbf{x}), G_2(\mathbf{x}), \ldots, G_n(\mathbf{x})) = (f_1(\mathbf{x}, k, c), f_2(\mathbf{x}, k, c), \ldots, f_n(\mathbf{x}, k, c))$$

resembles a parallel composition of PRGs. Although due to the lack of uniformity of the family $\{G_j\}_{j=1}^n$, Theorem 3.2 in [5] for the parallel composition of PRGs no longer holds (at least not directly since the PRGs G_j could be different). However, minor changes in the proof lead to the same results.

The Theorem 3.2 in [5] establishes a bound for the security of a parallel composition of PRGs by the sum over individual advantages for every component. The intuition behind that is to distinguish the sequence produced by G an adversary only needs to distinguish one component. The next Theorem deals with the non-uniformity of the family $\{G_j\}_{j=1}^n$ to state the same.

Theorem 1. *Let $G : \mathbb{F}^n \to \mathbb{F}^n$ be the mapping given by*

$$\mathbf{x} \mapsto (G_1(\mathbf{x}), G_2(\mathbf{x}), \ldots, G_n(\mathbf{x})) \tag{16}$$

Then, for every distinguishing adversary $\mathcal{A}$ over G there exists a distinguishing adversary $\mathcal{B}$ for G_j, for every $j \in \{1, \ldots, n\}$ such that

$$\mathrm{PRG}\,\mathtt{adv}(\mathcal{A}, G) = n \cdot \mathrm{PRG}\,\mathtt{adv}(\mathcal{B}, G_j) \tag{17}$$

Proof. Take any $j \in \{1, \ldots, n\}$. Let $\mathcal{A}$ be any adversary that interacts with the PRG G and try to differentiate samples from G from samples from the uniform distribution. We will build from $\mathcal{A}$ an adversary $\mathcal{B}$ that can distinguish the output of G_j. Briefly, this new adversary will pick a random value σ in $\{1, \ldots, n\}$ that plays the role of a

random coin. Then it will interact with $\mathcal{A}$ as a regular PRG challenger, passing back and forth the messages between $\mathcal{A}$ and the Challenger that uses G. At the end, $\mathcal{B}$ will output the answer of $\mathcal{A}$ only when the index of the component is j. If the index is different from j, it will output a random bit.

In detail, we define $\mathcal{B}$ with the following procedure:

1. Let $\sigma \xleftarrow{r} \{1, \ldots, n\}$.
2. Query to the PRG challenger that uses G. Let y be the answer.
3. Forward y to $\mathcal{A}$. Let b be the answer of $\mathcal{A}$
4. If $\sigma = j$, answer b. Else answer a random bit.

Following usual conventions, let Experiment 0 be the above procedure when the answer y is choosen uniformly at random from $\mathbb{F}^n$. Let Experiment 1 be the same procedure but with the answer y choosen from the output of G (as introduced in Section 2).

In the same way, Experiment 0 for the PRG G_j is the interaction with a Challenger using the PRG G_j that upon an adversaryt query, return a random element in $\mathbb{F}$. Experiment 1 for G_j on the other hand is the interaction with a Challenger using G_j that upon an adversary query, return the output of G_j.

If $W_b^{\mathcal{A}}$ is the event that $\mathcal{A}$ outputs 1 in Experiment b of the PRG G and W_b is the event that $\mathcal{B}$ outputs 1 in Experiment b of G_j, then provided $\sigma = j$, W_b happens if and only if $W_b^{\mathcal{A}}$ happens. That means

$$Pr[W_b \mid \sigma = j] = Pr[W_b^{\mathcal{A}}] \tag{18}$$

On the other hand, if $\sigma \neq j$ then $\mathcal{B}$ output 1 with probability $1/2$, so

$$Pr[W_1 \mid \sigma \neq j] = Pr[W_0 \mid \sigma \neq j] = 1/2. \tag{19}$$

From that, it follows easily

$$\begin{aligned}
\text{PRG}\mathtt{adv}(\mathcal{B}, G_j) &= |Pr[W_1] - Pr[W_0]| \\
&= |Pr[W_1 \mid \sigma \neq j]Pr[\sigma \neq j] - Pr[W_0 \mid \sigma \neq j]Pr[\sigma \neq j] \\
&\quad + Pr[W_1 \mid \sigma = j]Pr[\sigma = j] - Pr[W_0 \mid \sigma = j]Pr[\sigma = j]| \\
&= |Pr[W_1 \mid \sigma = j]Pr[\sigma = j] - Pr[W_0 \mid \sigma = j]Pr[\sigma = j]| \\
&= Pr[\sigma = j]|Pr[W_1 \mid \sigma = j] - Pr[W_0 \mid \sigma = j]| \\
&= \frac{1}{n}|Pr[W_1 \mid \sigma = j] - Pr[W_0 \mid \sigma = j]| \\
&= \frac{1}{n}\text{PRG}\mathtt{adv}(\mathcal{A}, G)
\end{aligned}$$

□

As the last step in this section, we will show a sequential PRG that uses state transitions. The procedure is a Blum-Micali iterative generator with the previous PRG G.

We will like to remind here the basic behaviour of the system described in Section **??**. Given a key k fixed for all the iterations, the system is started with a random initial state $\mathbf{x}_0$ which is updated to $\mathbf{x}_1$ with one application of $\mathcal{U}_k$ with the iteration parameter $c_0 = \mathcal{M}(\mathcal{O}(\mathbf{x}_0))$. So, the output of the system at $t = 1$ is $\mathcal{O}(\mathcal{U}_k(\mathbf{x}_0, c_0))$.

It is important to note that the design of $\mathcal{U}_k$ and $\mathcal{O}$ in the specific implementation must be carefully chosen to avoid $\mathcal{O} \circ \mathcal{U}_k$ being an easy-to-invert function.

Far from being a special requirement, this must be satisfied as well in the original stream cipher. The implication of not fulfilling that in the stream cipher is the exposure of subkeys while the implication in the PRG case is the predictability of the output sequence.

Algorithm 1 describes the generator.

Algorithm 1 Pseudo-random generator G^* :

Require: A seed $\mathbf{x}_0 \in \mathbb{F}^n$ (the initial state), a key $k \in \mathbb{F}^l$, an integer $s > 0$ (the number of random words to output)

1: $\mathbf{x} \leftarrow \mathbf{x}_0$ ▷ The current state
2: $r_0 \leftarrow \mathcal{O}(\mathbf{x})$ ▷ The first output
3: **for** $i = 0, \ldots, s-1$ **do**
4: $\quad c \leftarrow \mathcal{M}(r_i)$ ▷ The iteration parameter
5: $\quad \mathbf{x} \leftarrow \mathcal{U}_k(\mathbf{x}, c)$ ▷ The state transition
6: $\quad r_i \leftarrow \mathcal{O}(\mathbf{x})$ ▷ The output at iteration i
7: **end for**
8: **return** $r_0, \ldots, r_{s-1}$

Note $\mathcal{U}_k(\mathbf{x}, c) = (f_1(\mathbf{x}, k, c), \ldots, f_n(\mathbf{x}, k, c))$ so, line 5 in Algorithm 1 is an application of G with the key k and c as the iteration parameter. The security of the above PRG follows directly from that of G by sequential composition (see Theorem 3.3 in [5]). In particular, we have

Theorem 2. *For every adversary $\mathcal{A}$ for G^* there is an adversary $\mathcal{B}$ for G such that*

$$\mathrm{PRG}\boldsymbol{adv}(\mathcal{A}, G^*) = s \cdot \mathrm{PRG}\boldsymbol{adv}(\mathcal{B}, G)$$

The amount of bits produced per round is a very important point. We address that in the next section.

5 Hardness of the update and output functions composition

One of the key assumptions over the systems in [8] is the hardness of performing the inverse transition of a state, by only looking at the output ciphertext. Informally, any probabilistic polynomial time adversary $\mathcal{A}$ taking as its input the output $\mathbf{y} = \mathcal{O}(\mathcal{U}_k(\mathbf{x}, c))$, will only succeed in getting the original state $\mathbf{x}$ with negligible probability.

That will be called the *UO* assumption and can be formalized as follows:

Assumption 1 (UO assumption over the pair $\mathcal{U}_k, \mathcal{O}$)**.** *For a family of update functions* $\{\mathcal{U}_k : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}^n\}_{k \in \mathbb{F}^l}$ *and output functions* $\mathcal{O} : \mathbb{F}^n \to \mathbb{F}^p$*, with* $\mathbb{F}$ *a finite field of* λ *elements, and any key* $k \in \mathbb{F}^l$ *it holds that:*

For any probabilistic polynomial time adversary $\mathcal{A}$ *we have*

$$Pr\left[\begin{array}{c} \mathbf{x} \xleftarrow{r} \mathbb{F}^n; c \leftarrow \mathcal{M}(\mathcal{O}(\mathbf{x})); \\ \mathbf{y} \leftarrow \mathcal{O}(\mathcal{U}_k(\mathbf{x}, c)); \mathbf{x}' \leftarrow \mathcal{A}(y) : \\ \mathbf{x} = \mathbf{x}' \end{array}\right] = \mathtt{negl}(\lambda n)$$

The UO assumption is preserved under permutations in the state vector $\mathbf{x}$. That is, if P is a $n \times n$ permutation matrix, then getting $\mathbf{x}$ from $\mathbf{y}$ such that $\mathbf{y} = \mathcal{O}(\mathcal{U}_k(\mathbf{x}, c))$ is as hard as getting $\mathbf{x}$ from $\mathbf{y}'$ such that $\mathbf{y}' = \mathcal{O}(\mathcal{U}_k(P\mathbf{x}, c))$.

Proposition 1. *Let* P *be a permutation matrix of size* $n \times n$*. Let* $k \in \mathbb{F}^l$ *be an arbitrary key,* $\mathcal{U}_k : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}^n$ *an update function, and* $\mathcal{O} : \mathbb{F}^n \to \mathbb{F}^p$ *the output function. Let also,* $\mathcal{U}'_k$ *be an update function defined as* $\mathcal{U}'_k(x, c) = \mathcal{U}_k(P\mathbf{x}, c)$*. Then, for every adversary* $\mathcal{A}$ *inverting the composition* $\mathcal{O} \circ \mathcal{U}_k$ *there exists an adversary* $\mathcal{A}'$ *that inverts the composition* $\mathcal{O} \circ \mathcal{U}'_k$ *and both succeed with the same probability.*

Formally, we have that

$$Pr\left[\begin{array}{c} \mathbf{x} \xleftarrow{r} \mathbb{F}^n; c \leftarrow \mathcal{M}(\mathcal{O}(\mathbf{x})); \\ \mathbf{y} \leftarrow \mathcal{O}(\mathcal{U}_k(\mathbf{x}, c)); \mathbf{x}' \leftarrow \mathcal{A}(y) : \\ \mathbf{x} = \mathbf{x}' \end{array}\right] = Pr\left[\begin{array}{c} \mathbf{x} \xleftarrow{r} \mathbb{F}^n; c \leftarrow \mathcal{M}(\mathcal{O}(\mathbf{x})); \\ \mathbf{y} \leftarrow \mathcal{O}(\mathcal{U}'_k(\mathbf{x}, c)); \mathbf{x}' \leftarrow \mathcal{A}'(y) : \\ \mathbf{x} = \mathbf{x}' \end{array}\right]$$

Proof. The proof is a straightforward calculation. Let $\mathbf{y} = \mathcal{O}(\mathcal{U}_k(\mathbf{x}, c)))$ for some x and $c = \mathcal{M}(\mathcal{O}(\mathbf{x}))$. Then, we have that $\mathbf{y}$ is the output of the related system that uses update function $\mathcal{U}_k$ for the input argument $P^{-1}\mathbf{x}$. In detail:

$$\mathcal{O}(\mathcal{U}'_k(P^{-1}\mathbf{x}, c)) = \mathcal{O}(\mathcal{U}_k(P(P^{-1}\mathbf{x}, c))) = \mathcal{O}(\mathcal{U}_k(\mathbf{x}, c))) = \mathbf{y}.$$

So, given $\mathcal{A}$, to build $\mathcal{A}$' we only need to apply P^{-1} to the output of $\mathcal{A}$. Note that $\mathcal{A}'$ succeeds inverting y if and only if $\mathcal{A}$ succeeds inverting y. That proves the proposition.

□

Upper bound for the output

A fact easy to observe regarding the output function $\mathcal{O}$ is that it should be hard to invert. In particular, if $\mathcal{O}$ outputs p components of the state vector $\mathbf{x}$, that is

$$\mathcal{O}(\mathbf{x}) = (\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_p})$$

then the number of components that are not output should be greater or equal the number of elements of the key k (considered as a vector over $\mathbb{F}$).

As we pointed out before, knowing the initial state $\mathbf{x}$ and the update result $\mathbf{x}' = \mathcal{U}_k(\mathbf{x}, c)$ allows an adversary to recover the key k by solving a system of equations since $\mathbf{x}' = \mathbf{A}\mathbf{x}$ for some matrix $\mathbf{A} \in \mathbb{F}^{n \times n}$.

Leaving aside the complexity of solving the equations or the possibility that the adversary does not known the set of indexes $i_1, \ldots, i_p$ such that $\mathbf{y}_j = \mathbf{x}_{i_j}$, increasing the amount p of components output by $\mathcal{O}$ in such a way that $n - p$ is lower than the number of elements in the key k should be avoided as it can represent a serious security risk.

Having the key and the initial state, an adversary can completely run the system which implies predictability of the output. The point we want to make here is that the remaining bits in the internal state that are not used as output must be at least the same number of bits used to represent the key.

To put in a concrete way, let us denote $l = |I_1| + \ldots + |I_n|$ the number of elements in the key k. Then, the output size p should be such that $n - p \geq l$.

Unpredictability of hidden components

Proposition 1 shows that there are no special indexes in the state vector (as long we assume that the system is no easy to invert). Following that and the discussion on in the previous paragraph, we can expect that hidden exact l components (the same number of elements that the key has) will keep the generator secure. Although a formal proof of this should be conducted in future work, is important to note that this leads to a much more efficient generator. In fact, according to this observation output size could be increased from 1 to $n-l$ elements of the field. Since key size l is usually much smaller than n (leaving aside the toy example in Section 3), this means a significant enhancement in the output size.

6 Conclusions

We have shown here a family of functions suitable for implementing cryptographic primitives with provable randomness preservation. In addition, the functions in this family are built with only sboxes, multiplications, and additions in a finite field. The instance of functions in the family, either in hardware or software have low complexity and can be run fast, which makes all the family adequate for lightweight devices. That allows us to construct a novel PRG as iterative composition of the state update and output functions.

The entire design borrows components and ideas from the stream cipher given by [8], so in addition the work done here contributes to improve that family of stream ciphers.

Lastly, in Section **??**, we formalize the needed notions to increase the throughput of the PRG and their implications.

Statements and Declarations

Data availability. Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

References

[1] L'Ecuyer, P., Simard, R.: Testu01: A c library for empirical testing of random

number generators. ACM Trans. Math. Softw. **33**(4) (2007) https://doi.org/10.1145/1268776.1268777

[2] Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications. Technical report, NIST (April 2010). http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf

[3] Baksi, A., Breier, J., Chen, Y., Dong, X.: Machine learning assisted differential distinguishers for lightweight ciphers. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 176–181 (2021). https://doi.org/10.23919/DATE51398.2021.9474092

[4] Brunetta, C., Picazo-Sanchez, P.: Modelling cryptographic distinguishers using machine learning. Journal of Cryptographic Engineering **12**, 123–135 (2022) https://doi.org/10.1007/s13389-021-00262-x

[5] Boneh, D., Shoup, V.: A graduate course in applied cryptography. Springer (2023). http://toc.cryptobook.us/

[6] Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99, pp. 12–23. Springer, Berlin, Heidelberg (1999)

[7] Joux, A., Muller, F.: Loosening the knot. In: Johansson, T. (ed.) Fast Software Encryption, pp. 87–99. Springer, Berlin, Heidelberg (2003)

[8] Francq, J., Besson, L., Huynh, P., Guillot, P., Millerioux, G., Minier, M.: Non-triangular self-synchronizing stream ciphers. IEEE Transactions on Computers **71**(1), 134–145 (2022) https://doi.org/10.1109/TC.2020.3043714

[9] Patel, S., Sundaram, G.S.: An efficient discrete log pseudo random generator. In: Annual International Cryptology Conference (1998)