Andreeva *et al*. *J Surveill Secur Saf* 2024;5:36-51
DOI: 10.20517/jsss.2023.45

Journal of Surveillance, Security and Safety

Original Article

Open Access

A TPRF-based pseudo-random number generator

Elena Andreeva, Andreas Weninger

Institute of Logic and Computation, TU Wien, Karlsplatz 13, Vienna 1040, Austria.

Correspondence to: Andreas Weninger, Institute of Logic and Computation, TU Wien, Karlsplatz 13, Vienna 1040, Austria. E-mail: andreas.weninger@tuwien.ac.at

How to cite this article: Andreeva E, Weninger A. A TPRF-based pseudo-random number generator. *J Surveill Secur Saf* 2024;5:36-51. http://dx.doi.org/10.20517/jsss.2023.45

Received: 29 Nov 2023 **First Decision:** 29 Dec 2023 **Revised:** 9 Jan 2024 **Accepted:** 16 Jan 2024 **Published:** 28 Jan 2024

Academic Editor: Josef Pieprzyk **Copy Editor:** Yanbin Bai **Production Editor:** Yanbin Bai

Abstract

Most cryptographic applications use randomness that is generated by pseudo-random number generators (PRNGs). A popular PRNG practical choice is the NIST standardized CTR_DRBG. In their recent ACNS 2023 publication, Andreeva and Weninger proposed a new and more efficient and secure PRNG called FCRNG. FCRNG is based on CTR_DRBG and uses the n-to-$2n$ forkcipher expanding primitive ForkSkinny as a building block. In this work, we create a new BKRNG PRNG, which is based on FCRNG and employs the novel n-to-$8n$ expanding primitive Butterknife. Butterknife is based on the Deoxys tweakable blockcipher (and thus AES) and realizes a tweakable expanding pseudo-random function. While both blockciphers and forkciphers are invertible primitives, tweakable expanding pseudo-random functions are not. This functional simplification enables security benefits for BKRNG in the robustness security game - the standard security goal for a PRNG. Contrary to the security bound of CTR_DRBG, we show that the security of our BKRNG construction does not degrade with the length of the random inputs, nor the number of requested output pseudo-random bits. We also empirically verify the BKRNG security with the NIST PRNG test suite and the TestU01 suite.

Furthermore, we show the n-to-$8n$ *multi-branch* expanding nature of Butterknife contributes to a significant speed-up in the efficiency of BKRNG compared to FCRNG. More concretely, producing random bits with BKRNG is 30.0% faster than FCRNG and 49.2% faster than CTR_DRBG.

Keywords: Symmetric cryptography, pseudo-random number generation, tweakable PRF

INTRODUCTION

Good randomness is needed for almost all cryptographic protocols. Physical true randomness sampling devices usually do not produce perfectly uniform outputs, but more importantly, they are too slow for many practical applications. For this reason, pseudo-random number generators (PRNGs) are used. They expand high entropy bitstrings into longer bitstrings that are indistinguishable from uniform random. PRNGs are of high practical and theoretical interest.

Barak and Halevi[1] are the first to propose a theoretical model that defines how a PRNG recovers from a state compromise when it is given good randomness in the so-called refresh algorithm. Prior to that, PRNGs were treated as single functions that take a uniformly random seed as input to produce a single (long) output. In 2013, Dodis *et al.*[2] extended the model of Barak and Halevi so as to not require the PRNG to fully recover in a single call to the refresh algorithm. Instead, their model allows the PRNG to slowly accumulate randomness over the course of many calls to the refresh algorithm, which is the commonly accepted security model at present.

In terms of practical PRNGs, NIST recommends several PRNGs in their SP 800-90A standard. Among these, the AES-based `CTR_DRBG` is the most popular choice. (Cohney *et al.*[3] noted that 67.8% of all certified implementations from NIST's Cryptographic Module Validation Program (CMVP) in 2019 supported `CTR_DRBG`, making it the most popular design among these certifications.) Its security was formally analyzed in 2020 by Hoang and Shen[4]. The `CTR_DRBG` design includes some odd choices, such as running CBC-MAC three times for the randomness extraction algorithm (called CtE by Hoang and Shen[4]), without a clear justification for its provable security.

In 2023, Andreeva and Weninger[5] proposed a new PRNG called FCRNG. FCRNG is based on `CTR_DRBG` and improves the internal algorithms of `CTR_DRBG`. Furthermore, it replaces the use of a blockcipher as a building block with a forkcipher – a primitive introduced by Andreeva *et al.*[6]). FCRNG with forkcipher achieves better performance and security than `CTR_DRBG` and is better suited for lightweight applications. Forkciphers are similar to (tweakable) block ciphers, yet they produce two ciphertext blocks for a single input message block. Current forkciphers, such as ForkSkinny[6], are based on existing tweakable blockciphers, such as Skinny[7]. A forkcipher produces the output more efficiently than two evaluations of the corresponding tweakable block cipher. The expanding nature of forkciphers makes them a natural fit for PRNGs. Similar to PRNGs which expand by designing a short high entropy string into a longer pseudo-random output, forkciphers expand their input. As a result, the FCRNG forkcipher-based design is 33% faster than the block cipher-based `CTR_DRBG`[5]. FCRNG also offers improved security over `CTR_DRBG` due to the improved internal randomness extraction algorithm (FCTRCond) and the fact that the used internal forkcipher ForkSkinny is tweakable. The forkciphers are yet underused in FCRNG as they have functionalities that are not utilized by the FCRNG design. In particular, forkciphers are invertible (given one of the ciphertext blocks and the key, the message can be computed), and they offer the functionality to compute one ciphertext block from the other when given the key.

On the other hand, tweakable Pseudo-Random Functions (TPRF) are a type of expandable primitives that have only forward evaluating functionality. A recent instance of a TPRF is the Butterknife[8] construction. Butterknife offers eight-fold (n-to-$8n$-bit) input expansion as compared to the two-fold (n-to-$2n$-bit) expansion of forkciphers. At the same time, Butterknife prohibits inversion and computation of one output block from the other by design, with the added benefit of additionally eliminating extra attack vectors. Butterknife is based on Deoxys[9] and thus reuses internally the AES structure. Hence, it can utilize existing implementation optimizations, such as the Intel AES-NI. All these Butterknife features can be advantageous when it comes to generating pseudo-random outputs. Butterknife is already used for efficient key derivation in the Signal protocol[10]. Another natural Butterknife application is in generalized CTR mode encryption[11].

Contributions

In this work, we present our new BKRNG (ButterKnife PRNG) construction. The design idea of BKRNG is based on the FCRNG[5] and the NIST standardized CTR_DRBG constructions. The main difference is in the secure and efficient integration of a TPRF as an internal primitive. For our practical instantiation, we suggest the Butterknife TPRF. BKRNG conforms to the notion of a robust PRNG by Dodis *et al.*[2]; thus, it comprises a setup function for the initial setup, a refresh function for absorbing random inputs (to recover after a possible internal state compromise) and a next function for producing pseudo-random outputs.

- Our BKRNG construction is similar to FCRNG. FCRNG was specified with two variants, FCRNG-c and FCRNG-t. FCRNG-c gives the better performance, while FCRNG-t offers better security. This improvement in security was achieved by changing the tweak for each output block. This was needed to allow for a specific output block to repeat within the same call to the randomness generation function next. While Butterknife does offer tweakability as well, changing the tweak is not required since the individual output blocks are independent by design. For this reason, we chose to design BKRNG similar to the more performant FCRNG-c. By not changing the tweak for each output block, we allow practical implementations to compute the tweakey schedule only once instead of repeatedly refreshing it for each block thus further pushing the performance improvement.
- We provide full security proof for BKRNG in the robustness security game by Dodis *et al.*[2]. The proof bears similarity to the proofs for FCRNG[5] and CTR_DRBG[4]. Similar to the previous PRNG security proofs, we treat the internal TPRF primitive as ideal, which means that we consider it to be randomly drawn from all possible TPRFs with the same input, output, and tweakey size. Changing the internal primitive to a TPRF required new analysis since both forkciphers and blockciphers are invertible. This change allowed us to prove a security bound for BKRNG that is strictly better than those of CTR_DRBG and FCRNG (both for FCRNG-c and FCRNG-t). Compared to FCRNG-c (with which we also compare the performance in the next paragraph), the security bound of BKRNG improves several constant factors and entirely removes the summand $\frac{12pq}{2^n}$, where p is the number of queries to the setup and refresh algorithms and q the number of queries to next. Compared to the analysis of CTR_DRBG by Hoang and Shen[4], we additionally eliminated a summand related to the maximum length of each random input and a summand related to the maximum length of each pseudo-random output.
- We implement BKRNG and compare its performance to FCRNG and CTR_DRBG. Our results show that generating pseudo-random output with BKRNG is 30.0% faster than FCRNG (in its performance-oriented variation FCRNG-c) and 49.2% faster than CTR_DRBG. We also used our implementation to run the NIST PRNG test suite and the TestU01 suite and successfully passed all tests.

METHODS

Preliminaries

Notation.

Let $a + b$ and $a * b$ denote regular integer addition and multiplication, respectively. By $a \oplus b$, we denote bitwise XOR of two (equal length) bitstrings a, b. Let $\lceil r \rceil$ denote the smallest integer i s.t. $i \geq r$ for the real number r. $|a|$ denotes the length of bitstring a. By $[x]_y$, we denote encoding the value x as a bitstring of length y. $X[a : b]$ denotes the bitstring that is obtained by taking bits $a, a+1, \ldots, b$ of bitstring X. $a||b$ denotes the concatenation of bitstrings a, b. Let $M_1, \ldots, M_m \xleftarrow{n} M$ denote splitting a bitstring M, with $|M|$ being a multiple of n, into the blocks $M_1, \ldots, M_m$ $(\forall 1 \leq i \leq m : |M_i| = n$; hence, $m = |M|/n)$. $\mathsf{Perm}(n)$ denotes the set of all permutations with range and domain $\{0,1\}^n$.

Blockciphers and Forkciphers.

A blockcipher E is defined as the pair of algorithms (E, E^{-1}), where $E, E^{-1} : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$, and it holds that $\forall K \in \{0,1\}^k, M \in \{0,1\}^n : E^{-1}(K, E(K, M)) = M$. We denote by k the key size and n the block size. By $\mathsf{BC}(k, n)$, we denote the set of all such blockciphers.

Following[6], a forkcipher is a pair of deterministic algorithms, the forward encrypting and inverse algorithms, respectively:

$$F : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \times \{\mathtt{0},\mathtt{1},\mathtt{b}\} \to \{0,1\}^n \cup \{0,1\}^n \times \{0,1\}^n$$
$$F^{-1} : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \times \{0,1\} \times \{\mathtt{i},\mathtt{o},\mathtt{b}\} \to \{0,1\}^n \cup \{0,1\}^n \times \{0,1\}^n$$

Note that we use $\cup$ in the definitions since the output can be a single n-bit string or a pair of such strings, depending on the chosen mode, as we define below. A forkcipher uses an additional tweak of size t. By $\mathsf{FC}(k,t,n)$, we denote the set of all such forkciphers. A tweakable forkcipher F meets the *correctness condition* if for every $K \in \{0,1\}^k$, $\mathsf{T} \in \{0,1\}^t$, $M \in \{0,1\}^n$ and $\beta \in \{0,1\}$, all of the following conditions are met:

1. $\mathsf{F}^{-1}(K,\mathsf{T},\mathsf{F}(K,\mathsf{T},M,\beta),\beta,\mathtt{i}) = M$
2. $\mathsf{F}^{-1}(K,\mathsf{T},\mathsf{F}(K,\mathsf{T},M,\beta),\beta,\mathtt{o}) = \mathsf{F}(K,\mathsf{T},M,\beta\oplus 1)$
3. $(\mathsf{F}(K,\mathsf{T},M,\mathtt{0}),\mathsf{F}(K,\mathsf{T},M,\mathtt{1})) = \mathsf{F}(K,\mathsf{T},M,\mathtt{b})$
4. $(\mathsf{F}^{-1}(K,\mathsf{T},C,\beta,\mathtt{i}),\mathsf{F}^{-1}(K,\mathsf{T},C,\beta,\mathtt{o})) = \mathsf{F}^{-1}(K,\mathsf{T},C,\beta,\mathtt{b})$

For each pair of a key and a tweak, the forkcipher applies two independent permutations to the input to produce the two output blocks. We use the shorthand $F_K^{T,s}(m) := F(K,T,m,s)$. Since most of our algorithms only use $s = b$; we also use $F_K^T(m) := F_K^{T,\mathtt{b}}(m)$. Furthermore, denote $(F^{-1})_K^{T,\beta,s}(c) := \mathsf{F}^{-1}(K,T,c,\beta,s)$.

Almost Universal (AU) Hash.

Let $H : \mathsf{Seed} \times \mathsf{Dom} \to \{0,1\}^n$ be a (keyed) hash function. For each string X, define its *block length* to be $\max\{1, |X|/n\}$. For a function $\delta : \mathbb{N} \to [1,\infty)$, we say that H is a δ-*almost universal* hash if for every distinct strings X_1, X_2 whose block lengths are at most l, we have

$$\Pr_{seed \leftarrow\$ \mathsf{Seed}}[H(seed, X_1) = H(seed, X_2)] \le \frac{\delta(l)}{2^n}$$

Conditional Min-Entropy and Statistical Distance.

For two random variables X and Y, the *(average-case) conditional min-entropy* of X given Y is

$$\mathrm{H}_\infty(X|Y) = -\log\Big(\sum_y \Pr[Y=y] * \max_x \Pr[X=x|Y=y]\Big)$$

The *statistical distance* between two random variables X and Y is defined as

$$\mathsf{SD}(X,Y) = \frac{1}{2}\sum_z |\Pr[X=z] - \Pr[Y=z]|$$

$\mathsf{SD}(X,Y)$ is the best possible advantage of an (even computationally unbounded) adversary in distinguishing X and Y.

Systems, Transcripts and the H-coefficient Proof Technique.

Following[4,12], we consider the interactions of a distinguisher $\mathcal{A}$ with an abstract system S that answers $\mathcal{A}$'s queries. The resulting interaction then generates a transcript $\tau = ((X_1,Y_1),\ldots,(X_q,Y_q))$ of query-answer pairs. S is entirely described by the probabilities $\mathsf{ps}(\tau)$ that correspond to the system S responding with answers as indicated by τ when queries in τ are made. We will generally describe systems informally or more formally in terms of a set of oracles they provide and only use the fact that they define corresponding probabilities $\mathsf{ps}(\tau)$ without explicitly giving these probabilities. We say that a transcript is valid for system S if $\mathsf{ps}(\tau) > 0$.

For any systems S_1 and S_0, let $\Delta_{\mathcal{A}}(S_1, S_0)$ denote the distinguishing advantage of the adversary $\mathcal{A}$ against the "real" system S_1 and the "ideal" system S_0.

Following[4], we now describe the H-coefficient technique of Patarin[13,14]. Generically, it considers a deterministic distinguisher $\mathcal{A}$ that tries to distinguish a "real" system S_1 from an "ideal" system S_0. The adversary's interactions with those systems define transcripts X_1 and X_0, respectively, and a bound on the distinguishing advantage of $\mathcal{A}$ is given by the statistical distance $\mathsf{SD}(X_1, X_0)$.

Lemma 1 (see[13,14]). *Suppose we can partition the set of valid transcripts for the ideal system into good and bad ones. Further, suppose that there exists $\epsilon \geq 0$ such that $1 - \frac{\mathsf{ps}_1(\tau)}{\mathsf{ps}_0(\tau)} \leq \epsilon$ for every good transcript τ. Then,*

$$\mathsf{SD}(X_1, X_0) \leq \epsilon + \Pr[X_0 \textit{ is bad}]$$

Pseudo-Random Functions.
A TPRF is a keyed function $F : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \rightarrow \{0,1\}^m$.

Let $\mathsf{TPRF}(k, t, n, m)$ denote the set of all such TPRFs.

PRNG.
We recall the security definition by Dodis *et al.*[2]. A PRNG with input I with state space State and seed space Seed is a tuple of deterministic algorithms $G = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$.

- $\mathsf{setup}(seed, I)$ takes a seed $seed \in \mathsf{Seed}$ and a string I as input, to then output an initial state $S \in \mathsf{State}$.
- $\mathsf{refresh}(seed, S, I)$ takes as input $seed \in \mathsf{Seed}$, $S \in \mathsf{State}$, and string I and then outputs a new state.
- $\mathsf{next}(seed, S, l)$ takes as input $seed \in \mathsf{Seed}$, $S \in \mathsf{State}$, and a number $l \in \mathbb{N}$ to then output a new state and an l-bit output string.

For our constructions, we will not have an explicit seed but rather treat the full description of the idealized primitives (in our case, an ideal TPRF F) as the seed, as was done in several previous works (e.g.,[4,5]).

Condensers.
A condenser Cond[15] takes a bitstring that has low entropy (i.e., is not uniformly random) and outputs a value that is hard to predict. We follow[4] and[5] for the subsequent definitions. Let S be a λ-source, meaning a stateless, probabilistic algorithm that outputs a random input I and some side information z, such that $\mathrm{H}_\infty(I|z) \geq \lambda$. For any adversary $\mathcal{A}$, we define the guessing advantage of $\mathcal{A}$ against condenser Cond with a source S as

$$\mathsf{Adv}^{\mathrm{guess}}_{\mathsf{Cond}}(\mathcal{A}, S) = \Pr[\mathbf{G}^{\mathrm{guess}}_{\mathsf{Cond}}(\mathcal{A}, S)]$$

The corresponding game is described in Figure 1.

```
Game $\mathbf{G}^{\mathrm{guess}}_{\mathsf{Cond}}(\mathcal{A}, S)$
$(I, z) \leftarrow\$ S$; $seed \leftarrow\$ \mathsf{Seed}$; $V \leftarrow \mathsf{Cond}(seed, I)$
$(Y_1, \ldots, Y_q) \leftarrow\$ \mathcal{A}(seed, z)$; return $(V \in \{Y_1, \ldots, Y_q\})$

Game $\mathbf{G}^{\text{1-blk-guess}}_{\mathsf{Cond}}(\mathcal{A}, S)$
$(I, z) \leftarrow\$ S$; $seed \leftarrow\$ \mathsf{Seed}$; $V \leftarrow \mathsf{Cond}(seed, I)_{[1:n]}$
$(Y_1, \ldots, Y_q) \leftarrow\$ \mathcal{A}(seed, z)$; return $(V \in \{Y_1, \ldots, Y_q\})$
```

Figure 1. Security games for a condenser Cond

For any adversary $\mathcal{A}$, the 1-block-guessing advantage of $\mathcal{A}$ against condenser Cond with a source S is defined as

$$\mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{Cond}}(\mathcal{A}, S) = \Pr[\mathbf{G}^{\text{1-blk-guess}}_{\mathsf{Cond}}(\mathcal{A}, S)]$$

The corresponding security game is also described in Figure 1.

Distribution Samplers.

Distribution samplers are similar to λ-sources but are stateful and give an estimate of how much entropy they provide. A distribution sampler $\mathcal{D}$ is a stateful, probabilistic algorithm. Given the current state s, it will output a tuple (s', I, γ, z) in which s' is the updated state and I is the next randomness input for the PRNG G. $\gamma \geq 0$ is a real number that, informally speaking, will tell us the amount of entropy in I. z is some side information of I given to an adversary attacking G. Let p be an upper bound of the number of calls to $\mathcal{D}$ in our security games. Let s_0 be the empty string, and let $(s_i, I_i, \gamma_i, z_i) \leftarrow\!\!\$\ \mathcal{D}(s_{i-1})$ for every $i \in \{1, \ldots, p\}$. For each $i \leq p$, let

$$\mathcal{I}_{p,i} = (I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_p, \gamma_1, \ldots, \gamma_p, z_1, \ldots, z_p)$$

We say that sampler $\mathcal{D}$ is *legitimate* if $\mathrm{H}_\infty(I_i | \mathcal{I}_{p,i}) \geq \gamma_i$ for every $i \in \{1, \ldots, p\}$. A legitimate sampler is λ-simple if $\gamma_i \geq \lambda$ for every i. Following[4], in this work, we will only consider simple samplers for a sufficiently large min-entropy threshold λ. As they noted, this is somewhat limiting as it fails to show that the PRNG can slowly accumulate randomness by absorbing many low entropy inputs. However, the results are still meaningful and are the setting that was considered in the NIST SP 800-90A standard.

Robustness.

The game $\mathbf{G}^{\text{rob}}_{G,\lambda}(\mathcal{A}, \mathcal{D})$ is defined in Figure 2. It is played for a PRNG $G = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$, an adversary $\mathcal{A}$, and a distribution sampler $\mathcal{D}$, with respect to an entropy threshold λ. The internal variable c counts the current amount of entropy. While it is too low, the oracle RoR is designed to be useless to the adversary. Define

$$\mathsf{Adv}^{\text{rob}}_{G,\lambda}(\mathcal{A}, \mathcal{D}) = 2 * \Pr[\mathbf{G}^{\text{rob}}_{G,\lambda}(\mathcal{A}, \mathcal{D})] - 1$$

```
Game $\mathbf{G}^{\text{rob}}_{G,\lambda}(\mathcal{A}, \mathcal{D})$
$b \leftarrow\!\$ \{0,1\}; s \leftarrow \epsilon; seed \leftarrow\!\$ \mathsf{Seed}$
$c \leftarrow 0; (s, I, \gamma, z) \leftarrow\!\$ \mathcal{D}(s);$
$S \leftarrow \mathsf{setup}(seed, I); c \leftarrow c + \gamma$
$b' \leftarrow\!\$ \mathcal{A}^{\mathsf{REF},\mathsf{RoR},\mathsf{Get},\mathsf{Set}}(seed, \gamma, z)$
return $(b' = b)$
```

```
$\mathsf{REF}()$
$(s, I, \gamma, z) \leftarrow\!\$ \mathcal{D}(s)$
$S \leftarrow \mathsf{refresh}(seed, S, I); c \leftarrow c + \gamma$
return $(\gamma, z)$
```

```
$\mathsf{RoR}(1^l)$
$(R_1, S) \leftarrow \mathsf{next}(seed, S, l)$
if $(c < \lambda)$ then
    $c \leftarrow 0;$
    return $R_1$
$R_0 \leftarrow\!\$ \{0,1\}^l$; return $R_b$
```

```
$\mathsf{Get}()$
$c \leftarrow 0$
return $S$
```

```
$\mathsf{Set}(S^*)$
$S \leftarrow S^*; c \leftarrow 0$
```

Figure 2. Robustness game (see Fig. 1 in [4])

Generalized Leftover Hash Lemma

Lemma 2 (Generalized Leftover Hash Lemma[4]). *Let* $\mathsf{Cond} : \mathsf{Seed} \times \mathsf{Dom} \rightarrow \{0,1\}^n$ *be a* δ*-AU hash function, and let* $\lambda > 0$ *be a real number. Let* S *be a* λ*-source whose random input* I *has at most* l *blocks. For any adversary* $\mathcal{A}$*, making at most* q *guesses,*

$$\mathsf{Adv}^{guess}_{\mathsf{Cond}}(\mathcal{A}, S) \leq \frac{q}{2^n} + \sqrt{\frac{q}{2^\lambda} + \frac{q \cdot (\delta(l) - 1)}{2^n}}$$

Lemma 2 was first stated this way by Hoang and Shen[4] and was originally proven by Barak *et al.*[16]. We use the same security game $\mathbf{G}^{\text{guess}}_{\mathsf{Cond}}(\mathcal{A}, S)$ as[4]; thus, we can adopt this version of the notion.

Ideal TPRF

The ideal TPRF model is similar to the ideal cipher model: The TPRF F is modeled as being drawn uniformly randomly from $\mathsf{TPRF}(k, t, n, m)$. In security games, the adversary can query F directly with full control over all parameters.

Constructions

We present BKCond, our new condenser, and $\texttt{BKRNG}$, our new PRNG.

Condenser BKCond.

Our construction BKCond (Figure 3), a condenser, is based on $\mathsf{FCTRCond}$[5]. Instead of using a forkcipher as primitive, we use a TPRF $F \in \mathsf{TPRF}(k, t, n, m)$ with $m = 2n$.

As the main practical instantiation, we will use Butterknife[8] (restricted to 2 branches). With this instantiation, we have $k + t = 2n$; thus, the code in Figure 3 will not append any 0's at the end but discard the last bit instead.

```
$\mathsf{BKCond}(I)$
$M \leftarrow \mathsf{pad}^*(I)$
$M_1, ..., M_a \xleftarrow{k+n+v} M$
$B \leftarrow 0^{2n}$
for $i \leftarrow 1$ to $a$ do
    $K \leftarrow M_i[1 : k]$
    $V \leftarrow M_i[k + 1 : k + n]$
    $W \leftarrow 1 \,||\, [i]_{t-v-1} \,||$
        $M_i[k + n + 1 : k + n + v]$
    $B \leftarrow B \oplus F^W_K(V)$
if $|B| < k + t - 1$ then
    $B \leftarrow B \,||\, 0^{k+t-1-2n}$
return $B[1 : k + t - 1]$
```

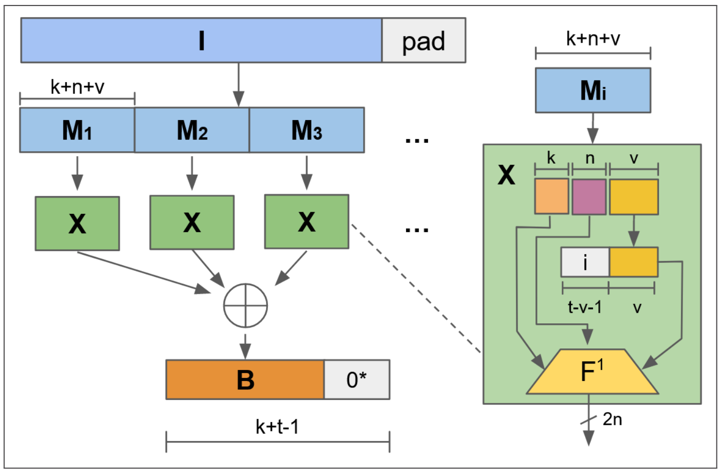

Figure 3. The condenser BKCond (see Figure 4 in[5]). F is a TPRF.

Our PRNG $\texttt{BKRNG}$.

In this section, we present our scheme $\texttt{BKRNG} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ (Figure 4), which is designed as a robust PRNG. It is based on FCRNG[5] and the NIST standardized $\texttt{CTR\_DRBG}$ (see analysis by[4]). Our construction is based on a TPRF F instead of a forkcipher (FCRNG) or a blockcipher (CTR-DRBG). As the main practical instantiation, we will use Butterknife[8].

```
$\mathsf{setup}^F(I)$
$X \leftarrow \mathsf{Cond}(\mathsf{I})$
$K \leftarrow 0^k; V \leftarrow 0^{t-1}$
$S \leftarrow \mathrm{PRFCTR}_K^V(X)$
return $S$
```

```
$\mathsf{refresh}^F(S, I)$
$X \leftarrow \mathsf{Cond}(\mathsf{I})$
$K \leftarrow S_{[1:k]}$
$V \leftarrow S_{[k+1:k+t-1]}$
$S \leftarrow \mathrm{PRFCTR}_K^V(X)$
return $S$
```

```
$\mathsf{next}^F(S, l)$
$K \leftarrow S_{[1:k]}; V \leftarrow S_{[k+1:k+t-1]}$
$r \leftarrow n * \lceil l/n \rceil$
$P \leftarrow \mathrm{PRFCTR}_K^V(0^{r+k+t-1})$
$R \leftarrow P_{[1:l]}$
$S \leftarrow P_{[r+1:r+k+t-1]}$
return $(R, S)$
```

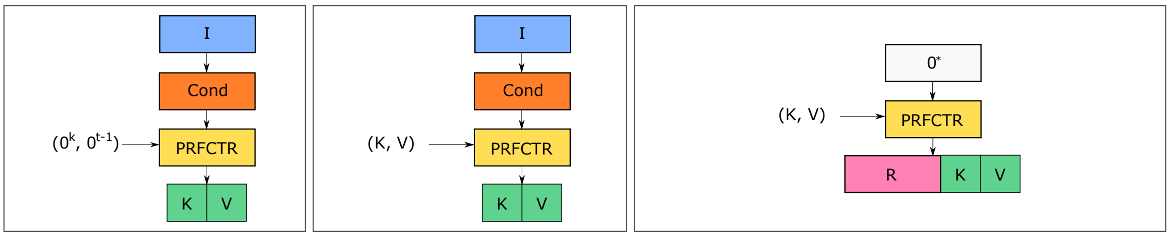

Figure 4. The construction BKRNG. Cond denotes a condenser, and F is a TPRF. PRFCTR is described in Figure 5.

```
$\mathrm{PRFCTR}_K^V(M)$
$P \leftarrow \emptyset$    // empty string
$\mathsf{ctr} \leftarrow 0$
while $|P| < |M|$ do
    $T = 0||V$
    $P = P||F_K^T([\mathsf{ctr}]_n)$
    $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
$C \leftarrow M \oplus P[1 : |M|]$
return $C$
```

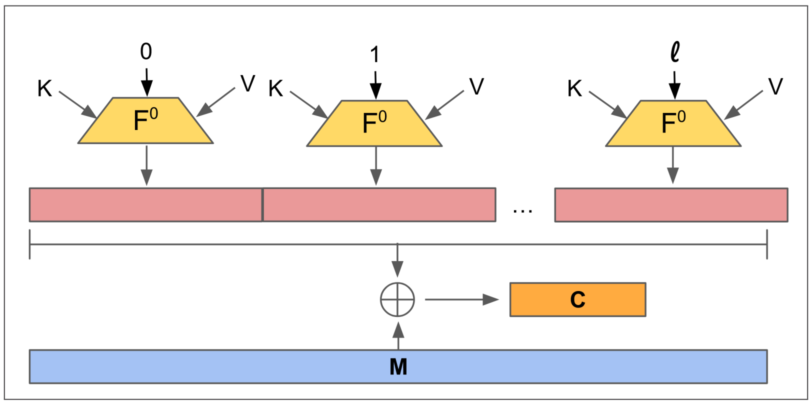

Figure 5. Algorithm PRFCTR (tPRF CounTeR mode). F is a TPRF. The tweak starts with the 0 bit for domain separation with respect to calls in BKCond.

Security Proofs

We now give a security proof for BKCond, a generic proof for BKRNG, and the security when instantiated with BKCond.

Security of BKCond

Theorem 1. *Let F be a TPRF that we model as an ideal TPRF. Let* BKCond *be as described in Figure 3. Let S be a λ-source that does not have access to F. Then, for any adversary $\mathcal{A}$ against* BKCond *in the 1-block-guessing game, making at most q guesses has an advantage at most.*

$$\mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{BKCond}}(\mathcal{A}, S) \leq \frac{q}{2^n} + \frac{\sqrt{q}}{2^{\lambda/2}}$$

To prove this theorem, we will show that BKCond is a good AU-hash. Let BKCond^* the construction that always returns the first block BKCond, i.e., $\mathsf{BKCond}^*(I) = \mathsf{BKCond}(I)_{[1:n]}$.

Lemma 3. *Let F be a TPRF that we model as an ideal TPRF. Let BKCond^* be as described above. Let I_1, I_2 be arbitrary strings s.t. $I_1 \neq I_2$. Then, $\Pr[\mathsf{BKCond}^*(I_1) = \mathsf{BKCond}^*(I_2)] \leq \frac{1}{2^n}$ where the randomness is taken over the choices of F.*

Proof. Let $Y_1, \dots, Y_y$ be the random variables corresponding to the first blocks of the outputs of the F-calls when computing $\mathsf{BKCond}^*(I_1)$. (Thus, $\mathsf{BKCond}^*(I_1) = Y_1 \oplus \dots \oplus Y_y$.) Analogously, let $Z_1, \dots, Z_z$ correspond to the blocks for computing $\mathsf{BKCond}^*(I_2)$. Since $I_1 \neq I_2$, there is at least one block in I_1 or I_2, which is not present in the other string at that position. Without loss of generality, assume it is the first block of I_1. Let $\alpha_{y_2,\dots,y_y,z_1,\dots,z_z}$ denote $Y_2 = y_2, \dots, Y_y = y_y, Z_1 = z_1, \dots, Z_z = z_z$.

$$\begin{aligned}
&\Pr[\mathsf{BKCond}^*(I_1) = \mathsf{BKCond}^*(I_2)] \\
&= \sum_{y_2,\dots,y_y,z_1,\dots,z_z \in \{0,1\}^n} \Pr[Y_1 = y_2 \oplus \dots \oplus y_y \oplus z_1 \oplus \dots \oplus z_z | \alpha_{y_2,\dots,y_y,z_1,\dots,z_z}] \Pr[\alpha_{y_2,\dots,y_y,z_1,\dots,z_z}] \\
&= \sum_{y_2,\dots,y_y,z_1,\dots,z_z \in \{0,1\}^n} \frac{1}{2^n} \Pr[\alpha_{y_2,\dots,y_y,z_1,\dots,z_z}] = \frac{1}{2^n}
\end{aligned}$$

The first equality is due to the law of total probability. The second one follows from the fact that F is called only once with the inputs corresponding to Y_1 (due to the counter value in the tweak, and the two strings being different), and thus an independently randomly sampled value. □

As was done in previous works (e.g.,[4,5]), we treat the full description of the ideal F as equivalent to a seed. Therefore, Lemma 3 means that BKCond^* can be viewed as an δ-almost universal hash function, with $\delta(l) = 1$ for all $l \in \mathbb{N}$.

Lemma 4. *Let F be a TPRF that we model as an ideal TPRF. Let BKCond^* be as described above. Let S be a λ-source that does not have access to F. Then, for any adversary $\mathcal{A}$ against BKCond^* in the guessing game, making at most q guesses has an advantage at most.*

$$\mathsf{Adv}^{guess}_{\mathsf{BKCond}^*}(\mathcal{A}, S) \leq \frac{q}{2^n} + \frac{\sqrt{q}}{2^{\lambda/2}}$$

Proof. We use Lemma 2 and apply the result of Lemma 3 (thus $\delta(l) = 1$). □

From the above lemma, we can immediately derive Theorem 1.

Security of BKRNG

We will now prove the security of BKRNG. Specifically, we will show that it is a robust PRNG. Our proof strategy is similar to that of Andreeva and Weninger[5] and Hoang and Shen[4], with the main difference being that the internal primitive is a TPRF in our case (instead of a forkcipher or blockcipher). This means we can avoid classifying a certain set of transcripts as bad in terms of the H-coefficient technique, thus improving security.

First, we define some parameters. We model the TPRF F underlying our construction as ideal. The adversary $\mathcal{A}$ is allowed to make at most q oracle queries, for which they can query F and REF, RoR, Get, Set (see Figure 2). Let $\mathcal{D}$ be a λ-simple distribution sampler. The number p denotes the maximum number of random inputs I that are produced by $\mathcal{D}$. (Such random inputs are produced in the initial setup of the game $\mathsf{Adv}^{\mathrm{rob}}_{G,\lambda}(\mathcal{A}, \mathcal{D})$ and the calls to REF.) l_i denotes the maximum block length of the i-th random input produced by $\mathcal{D}$.

Theorem 2. *Let F be a TPRF that we model as an ideal TPRF. Let Cond be a condenser without access to F and let $\mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{Cond}}(q', l')$ denote the maximum advantage against Cond of any adversary making at most q' queries, where l' is the maximum block length of the random inputs to Cond. Let BKRNG be the construction as defined in Figure 4. Let $\mathcal{D}$ be a λ-simple distribution sampler and $\mathcal{A}$ be an attacker against the robustness of*

BKRNG whose accounting of queries is given above. Then

$$\mathsf{Adv}^{\mathrm{rob}}_{BKRNG,\lambda}(\mathcal{A},\mathcal{D}) \leq \frac{4q^2}{2^k} + 4\sum_{j=1}^{p} \mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{Cond}}(q,l_j).$$

Proof. In our model, we consider computationally unbounded adversaries. For this reason, we are able to treat the adversary $\mathcal{A}$ as deterministic without loss of generality. Let $\mathsf{S}_{\mathsf{ideal}}$ and $\mathsf{S}_{\mathsf{real}}$ be the systems that model the oracles accessed by $\mathcal{A}$ in $\mathbf{G}^{\mathrm{rob}}_{G,\lambda}(\mathcal{A},\mathcal{D})$ with challenge bit $b = 0$ and $b = 1$, respectively.

For our hybrid argument, we create an intermediate game, $\mathsf{S}_{\mathsf{hybrid}}$. It behaves similarly to $\mathsf{S}_{\mathsf{real}}$ but with some changes. Firstly, when running PRFCTR, instead of using the underlying TPRF F, it produces uniformly random bitstrings of length m (i.e., the output length of F). Thus, the output of such a PRFCTR call is also uniformly random. However, when the min-entropy level c is lower than the threshold λ, $\mathsf{S}_{\mathsf{hybrid}}$ will still run the original PRFCTR. This is done to prevent trivial attacks. As another change, $\mathsf{S}_{\mathsf{hybrid}}$ will keep track of the keys that occur in a list called Keys. The resulting pseudocode is shown in Figure 6, together with a similar modification that we apply to $\mathsf{S}_{\mathsf{real}}$ (the algorithm is functionally unchanged). We write $\mathsf{Keys}(S)$ with $S \in \left\{\mathsf{S}_{\mathsf{real}}, \mathsf{S}_{\mathsf{hybrid}}\right\}$ to denote the corresponding list of system S.

When looking at $\mathsf{S}_{\mathsf{hybrid}}$ and $\mathsf{S}_{\mathsf{ideal}}$, one might assume that they are trivially indistinguishable. This is not necessarily the case since $\mathsf{S}_{\mathsf{ideal}}$ only idealizes the output of next through the RoR oracle, whereas the updated PRFCTR in $\mathsf{S}_{\mathsf{hybrid}}$ influences the other algorithms of the PRNG as well (setup and $\mathsf{refresh}$). (Recall that $\mathcal{A}$ can obtain the internal state with Get.)

```
PRFCTR$_K^V$(M) in $\mathsf{S}_{\mathsf{real}}$
$O \leftarrow \epsilon$    // empty string
$\mathsf{ctr} \leftarrow 0$
if $c \geq \lambda$ then $\mathsf{Keys} \leftarrow \mathsf{Keys} \cup \{K\}$
while $|O| < |M|$ do
    $P = F_K^V([\mathsf{ctr}]_n)$
    $O = O||P$
    $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
return $M \oplus O[1 : |M|]$
```

```
PRFCTR$_K^V$(M) in $\mathsf{S}_{\mathsf{hybrid}}$
$O \leftarrow \epsilon$    // empty string
$\mathsf{ctr} \leftarrow 0$
if $c \geq \lambda$ then $\mathsf{Keys} \leftarrow \mathsf{Keys} \cup \{K\}$
while $|O| < |M|$ do
    if $c \geq \lambda$ then
        $P \leftarrow\$ \{0,1\}^m$
    else
        $P = F_K^V([\mathsf{ctr}]_n)$
    $O = O||P$
    $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
return $M \oplus O[1 : |M|]$
```

Figure 6. Updated PRFCTR Algorithm in security proof of BKRNG

Proof argument. We define four main steps as propositions and prove them afterward.

1. There is an adversary $\mathcal{A}^*$ s.t.

$$\Delta_{\mathcal{A}^*}(\mathsf{S}_{\mathsf{real}}, \mathsf{S}_{\mathsf{hybrid}}) = \Delta_{\mathcal{A}}(\mathsf{S}_{\mathsf{ideal}}, \mathsf{S}_{\mathsf{hybrid}})$$

2.

$$\Delta_{\mathcal{A}}(\mathsf{S}_{\mathsf{real}}, \mathsf{S}_{\mathsf{hybrid}}) \leq \frac{2q^2}{2^k} + 2\sum_{j=1}^{p} \mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{Cond}}(q,l_j)$$

3.

$$\Delta_{\mathcal{A}^*}(\mathsf{S}_{\mathsf{real}}, \mathsf{S}_{\mathsf{hybrid}}) \le \frac{2q^2}{2^k} + 2\sum_{j=1}^{p} \mathsf{Adv}_{\mathsf{Cond}}^{\text{1-blk-guess}}(q, l_j)$$

4.

$$\mathsf{Adv}_{\mathrm{BKRNG},\lambda}^{\mathrm{rob}}(\mathcal{A}, \mathcal{D}) \le \Delta_{\mathcal{A}}(\mathsf{S}_{\mathsf{real}}, \mathsf{S}_{\mathsf{hybrid}}) + \Delta_{\mathcal{A}^*}(\mathsf{S}_{\mathsf{real}}, \mathsf{S}_{\mathsf{hybrid}}) \tag{1}$$

From the above arguments, it follows that

$$\mathsf{Adv}_{\mathrm{BKRNG},\lambda}^{\mathrm{rob}}(\mathcal{A}, \mathcal{D}) \le \frac{4q^2}{2^k} + 4\sum_{j=1}^{p} \mathsf{Adv}_{\mathsf{Cond}}^{\text{1-blk-guess}}(q, l_j)$$

Proof of Item 1. We define $\mathcal{A}^*$ as follows. $\mathcal{A}^*$ runs $\mathcal{A}$ and uses its own oracles to answer $\mathcal{A}$'s oracle queries. Whenever $\mathcal{A}$ queries RoR, $\mathcal{A}^*$ responds with a uniformly random string if $c \ge \lambda$. When $\mathcal{A}$ outputs its final guess b', $\mathcal{A}^*$ also outputs b'. As a result, if $\mathcal{A}^*$ is in the real world, then it perfectly simulates $\mathsf{S}_{\mathsf{ideal}}$ for $\mathcal{A}$. On the other hand, if $\mathcal{A}^*$ interacts with $\mathsf{S}_{\mathsf{hybrid}}$, then $\mathcal{A}^*$ perfectly simulates $\mathsf{S}_{\mathsf{hybrid}}$ for $\mathcal{A}$.

Proof of bound on $\Delta_{\mathcal{A}}(\mathsf{S}_{\mathsf{real}}, \mathsf{S}_{\mathsf{hybrid}})$ (Item 2): Defining bad transcripts. We will prove this proposition by using the H-coefficient technique.

A transcript is called *bad* if one of the following conditions happens:

1. The transcript contains a query of $\mathcal{A}$ to F or F^{-1} using some key $K \in \mathsf{Keys}(\mathsf{S})$. In other words, $\mathcal{A}$ was able to guess or derive K.
2. There are two identical keys in $\mathsf{Keys}(\mathsf{S})$.

We do not need a Bad event that relates keys used in $\mathtt{PRFCTR}$ with keys used by Cond since Cond is required to be independent of F. In comparison to the security analysis for similar constructions[4,5], we were able to avoid several bad cases since our real construction is more similar to the ideal one. The reason is that we use the TPRF primitive that gives us general functions instead of permutations (as would be the case for blockciphers/forkciphers[4,5]).

If a transcript is not bad, then we say that it is good. Let $\mathcal{T}_{\mathsf{real}}$ and $\mathcal{T}_{\mathsf{hybrid}}$ be the random variable of the transcript for $\mathsf{S}_{\mathsf{real}}$ and $\mathsf{S}_{\mathsf{hybrid}}$, respectively.

Proof of bound on $\Delta_{\mathcal{A}}(\mathsf{S}_{\mathsf{real}}, \mathsf{S}_{\mathsf{hybrid}})$: Probability of bad transcripts. We continue by establishing a bound on the chance that $\mathcal{T}_{\mathsf{hybrid}}$ is bad. Let Bad_i be the event that $\mathcal{T}_{\mathsf{hybrid}}$ violates the i-th condition. By the union bound,

$$\Pr[\mathcal{T}_{\mathsf{hybrid}} \text{ is bad}] = \Pr[\mathsf{Bad}_1 \cup \mathsf{Bad}_2] \le \Pr[\mathsf{Bad}_1] + \Pr[\mathsf{Bad}_2]$$

We will start by giving a bound on $\Pr[\mathsf{Bad}_1]$. The keys in $\mathsf{Keys}(\mathsf{S}_{\mathsf{hybrid}})$ were put there during a call to $\mathtt{PRFCTR}$. They can be categorized as follows:

- **Idealized Keys:** The key was picked uniformly at random. (This is the case if there was enough entropy c during the previous call to $\mathtt{PRFCTR}$.)
- **Normal Keys:** This key is the result of

$$K_i \leftarrow \mathtt{PRFCTR}_{K_{i-1}}^{V_{i-1}}(\mathsf{Cond}(I))_{[1:k]}$$

(Indeed, all keys in $\mathsf{Keys}(\mathsf{S}_{\mathsf{hybrid}})$ that are not idealized must have been derived as described, and in particular, cannot be the result of a call to next. Since $K_i \in \mathsf{Keys}(\mathsf{S}_{\mathsf{hybrid}})$, we know that $c \ge \lambda$ in the

PRFCTR call that had K_i as key. Furthermore, we know that in the PRFCTR call before that $c < \lambda$, since K_i is not uniformly random. Because of the fact that c increased, there must have been a call to REF and, hence, Cond.)

For the idealized keys, the adversary has a probability of $q/2^k$ to guess a key with a single attempt. The adversary queries F at most q times (in other words q guesses), thus resulting in the probability of less than $q^2/2^k$ for the adversary to guess an idealized key.

For each $j \leq p$, let $\mathsf{Hit}_1(j)$ be the event that the key derived from the random input I_j is a normal key, and causes Bad_1 to happen. From the union bound

$$\Pr[\mathsf{Bad}_1] \leq \frac{q^2}{2^k} + \Pr[\mathsf{Hit}_1(1) \cup \ldots \cup \mathsf{Hit}_1(p)] \leq \frac{q^2}{2^k} + \sum_{j=1}^{p} \Pr[\mathsf{Hit}_1(j)].$$

For all $\mathsf{Hit}_1(j)$, we know K_j was derived during a REF query, which does not provide any information to $\mathcal{A}$ besides γ and z from $\mathcal{D}$. Then, the adversary has the following options for the next query: (a) corrupt the state using Get or Set, or (b) use REF or RoR. If (a) is used, then $c \leftarrow 0$. We can rule out this possibility since K_j is only added to Keys if $c \geq \lambda$. Attempting to use REF to increase c also overrides K_j before it is added to the list. In case (b), note that $c \geq \lambda$ during this next query since we only consider λ-simple distribution samplers. Hence, K_j is not being used at all and is immediately replaced with a new uniformly random key.

Thus, the only information available to $\mathcal{A}$ for guessing K_j is (γ, z). Guessing K_j implies guessing the first block of K_j, which is exactly the setting of $\mathbf{G}^{\text{1-blk-guess}}_{\mathsf{Cond}}(\mathcal{A}, S)$.

$$\Pr[\mathsf{Hit}_1(j)] \leq \mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{Cond}}(q, l_j)$$

Therefore, by summing up overall $\mathsf{Hit}_1(j)$, we obtain

$$\Pr[\mathsf{Bad}_1] \leq \frac{q^2}{2^k} + \sum_{j=1}^{p} \mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{Cond}}(q, l_j).$$

We continue by analyzing the probability of Bad_2. First, consider any idealized key colliding with any other key. The probability for the idealized key to collide with any of the other q keys in the system is at most $q/2^k$. Since there are at most q such keys, the probability of this happening is at most $q^2/2^k$. What remains is the case of two normal keys colliding. For any given normal key, the probability that another normal key is the same is bounded by $\mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{Cond}}(\mathcal{A}, S)$, similar to the argument regarding Bad_1. The only difference is that instead of the adversary directly guessing the outcome of $\mathsf{Cond}(I)$, the environment "guesses" the outcome. This results in the same bound;

$$\Pr[\mathsf{Bad}_2] \leq \frac{q^2}{2^k} + \sum_{j=1}^{p} \mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{Cond}}(q, l_j)$$

Summing up, we have

$$\Pr[\mathcal{T}_{\mathsf{hybrid}} \text{ is bad}] \leq \frac{2q^2}{2^k} + 2\sum_{j=1}^{p} \mathsf{Adv}^{\text{1-blk-guess}}_{\mathsf{Cond}}(q, l_j) \tag{2}$$

Proof of bound on $\Delta_{\mathcal{A}}(\mathsf{S}_{\text{real}}, \mathsf{S}_{\text{hybrid}})$: Transcript ratio. Let τ be a good transcript s.t. $\Pr[\mathcal{T}_{\text{hybrid}} = \tau] \geq 0$. We now prove that

$$1 - \frac{\Pr[\mathcal{T}_{\text{real}} = \tau]}{\Pr[\mathcal{T}_{\text{hybrid}} = \tau]} \leq 0 \tag{3}$$

Note that both the adversary $\mathcal{A}$ and the original game environment are deterministic. As such, all randomness of the transcript is determined by the randomness of the distribution sampler $\mathcal{D}$, the random instantiation of F, and the random values that $\mathsf{S}_{\text{hybrid}}$ produces in the modified PRFCTR (instead of querying F).

Thus, we can characterize the relevant probabilities as follows:

$$\begin{aligned}\Pr[\mathcal{T}_{\text{real}} = \tau] &= \Pr[\mathsf{Inputs}] \cdot \Pr[\mathsf{Prim}|\mathsf{Inputs}] \cdot \Pr[\mathsf{Coll}_{\text{real}}|\mathsf{Inputs} \cap \mathsf{Prim}] \\ \Pr[\mathcal{T}_{\text{hybrid}} = \tau] &= \Pr[\mathsf{Inputs}] \cdot \Pr[\mathsf{Prim}|\mathsf{Inputs}] \cdot \Pr[\mathsf{Coll}_{\text{hybrid}}|\mathsf{Inputs} \cap \mathsf{Prim}]\end{aligned} \tag{4}$$

where

- Inputs denote the event that the distribution sampler samples the same values as was done for τ.
- Prim denotes the event that the primitive F agrees with the result of any direct query to F and that it produces the correct values for PRFCTR calls where $c < \lambda$.
- $\mathsf{Coll}_{\text{real}}$ denotes the event that, in S_{real}, the randomly chosen F produces the correct values for PRFCTR calls where $c \geq \lambda$.
- $\mathsf{Coll}_{\text{hybrid}}$ denotes the event that, in $\mathsf{S}_{\text{hybrid}}$, the randomly chosen values in the modified PRFCTR comply with the transcript (i.e., when $c \geq \lambda$).

It follows

$$\frac{\Pr[\mathcal{T}_{\text{real}} = \tau]}{\Pr[\mathcal{T}_{\text{hybrid}} = \tau]} = \frac{\Pr[\mathsf{Coll}_{\text{real}}|\mathsf{Inputs} \cap \mathsf{Prim}]}{\Pr[\mathsf{Coll}_{\text{hybrid}}|\mathsf{Inputs} \cap \mathsf{Prim}]}.$$

Let Q be the total number of calls to F of queries for which $c \geq \lambda$ was the case. In $\mathsf{S}_{\text{hybrid}}$, these queries are answered in a uniformly (and independently) random way. Thus

$$\Pr[\mathsf{Coll}_{\text{hybrid}}|\mathsf{Inputs} \cap \mathsf{Prim}] = \frac{1}{(2^m)^Q}$$

Next, we examine $\mathsf{Coll}_{\text{real}}$. Due to the definition of a good transcript, we know that there are no two calls to F with the same key and message. Hence

$$\Pr[\mathsf{Coll}_{\text{real}}|\mathsf{Inputs} \cap \mathsf{Prim}] \leq \frac{1}{(2^m)^Q}$$

This proves Equation (3).

Wrapping it up. Finally, from Lemma 1, together with Equation (2) and Equation (3), it follows that

$$\Delta_{\mathcal{A}}(\mathsf{S}_{\text{real}}, \mathsf{S}_{\text{hybrid}}) \leq \frac{2q^2}{2^k} + 2\sum_{j=1}^{p} \mathsf{Adv}_{\mathsf{Cond}}^{\text{1-blk-guess}}(q, l_j)$$

Proof of Item 3. This follows immediately from Item 2 since the latter applies to all adversaries, and hence, it applies to adversary $\mathcal{A}^*$ as well.

Proof of Item 4. By the triangle inequality,

$$\begin{aligned}\mathsf{Adv}^{\text{rob}}_{\text{BKRNG},\lambda}(\mathcal{A},\mathcal{D}) &= \Delta_{\mathcal{A}}(\mathsf{S}_{\text{real}},\mathsf{S}_{\text{ideal}})\\ &\le \Delta_{\mathcal{A}}(\mathsf{S}_{\text{real}},\mathsf{S}_{\text{hybrid}}) + \Delta_{\mathcal{A}}(\mathsf{S}_{\text{hybrid}},\mathsf{S}_{\text{ideal}})\\ &= \Delta_{\mathcal{A}}(\mathsf{S}_{\text{real}},\mathsf{S}_{\text{hybrid}}) + \Delta_{\mathcal{A}^*}(\mathsf{S}_{\text{real}},\mathsf{S}_{\text{hybrid}})\end{aligned} \tag{5}$$

□

Theorem 3. *Let F be a TPRF that we model as an ideal TPRF. Let (BKRNG, BKCond) denote BKRNG where Cond is instantiated with BKCond. Let $\mathcal{D}$ be a λ-simple distribution sampler and $\mathcal{A}$ be an adversary attacking this construction whose accounting of queries is given above. Then*

$$\mathsf{Adv}^{\text{rob}}_{(\mathit{BKRNG},\mathsf{BKCond}),\lambda}(\mathcal{A},\mathcal{D}) \le \frac{4q^2}{2^k} + \frac{4pq}{2^n} + \frac{4p\sqrt{q}}{2^{\lambda/2}}$$

Proof sketch. This theorem can be proven in the same way as Theorem 2. Even though BKRNG and BKCond use the same TPRF F, the domain separation (i.e., prepending all tweaks with 0 or 1) makes it effectively two independent TPRFs. Thus, we can use the result from Theorem 1.

RESULTS AND DISCUSSION

Security

The security bound for BKRNG is strictly better than the designs it is based on, i.e., CTR_DRBG[4] and FCRNG[5] (both for FCRNG-c and FCRNG-t). Compared to FCRNG-c (with which we also compare the performance in the next paragraph), the security bound of BKRNG improves several constant factors and entirely removes the summand $\frac{12pq}{2^n}$, where p is the number of queries to the setup and refresh algorithms, and q is the number of queries to next. Compared to the analysis of CTR_DRBG by Hoang and Shen[4], we additionally eliminated a summand related to the maximum length of each random input and a summand related to the maximum length of each pseudo-random output. We also empirically verified the security of BKRNG using the NIST PRNG test suite as well as the TestU01 suite. BKRNG passed all tests.

Performance

We implemented BKRNG, FCRNG, and CTR_DRBG in C (without any platform-specific optimizations) and compared their performance. For the TPRF in BKRNG, we chose the Butterknife implementation by Simon Müller[17]. For AES in CTR_DRBG, we used the TinyAES implementation[18], and for ForkSkinny in FCRNG, we used an implementation by Erik Pohle[19]. The benchmarks were performed on a 64-bit machine with four CPUs (AMD EPYC 7713 64-Core Processor) and 4 GB of memory that runs Ubuntu 22.04 LTS. Table 1 shows that BKRNG, when generating pseudo-random numbers, performs 30.0% better than FCRNG and 49.2% better than CTR_DRBG (see also Figure 7). The setup and refresh algorithms are the fastest for FCRNG-c, followed by BKRNG and then CTR_DRBG. These algorithms have little impact on the overall performance of the algorithm since they can be executed during idle times. Even if this is not done, their impact is negligible compared to the cost of next for most applications (How often an application should refresh depends on the threat model, in particular how often an adversary is assumed to learn something about the internal state, e.g., through side-channels. If this is not a concern, one finds that the NIST SP 800-90A specification allows $q = 2^{45}$ next queries without reseeding for CTR_DRBG. For our construction with improved security, each of these queries can even be done for the maximum possible amount of requested bits, i.e., around 2^n blocks of $m = 8n$ bits.).

Table 1. Runtimes of different PRNG implementations in CPU cycles. Setup and refresh were called with 24 bytes; next was called to produce 1000000 bytes. The final column lists the cycles per produced byte

	setup	refresh	next	next (c/b)
BKRNG	89372	82166	277957107	277.96
FCRNG-c	44668	34646	396987185	396.99
CTR_DRBG	134121	126986	546715525	546.72

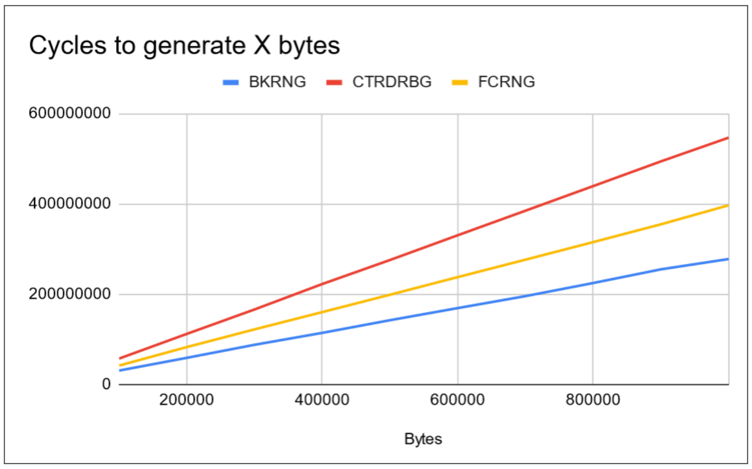

Figure 7. Comparison of PRNG performances. The requested output size ranged from 100000 to 1000000 bytes (depicted in the x-axis).

Example Output

When initializing the PRNG using the `setup` function on the following input, and then calling `next` to produce ten bytes of output results in the value given in Table 2.

Table 2. Example input and output (all values given in hexadecimal format)

Input	01 02 03 04 05 06 07 08
Output	70 af ef e9 6f d4 03 ee 10 78

DECLARATIONS

Acknowledgments

The authors thank the reviewers and the editors for their comments towards improving the article.

Authors' contributions

Designed the scheme, provided the proof and performed the experimental and theoretical evaluations: Weninger A

Started the project with the initial idea, gave technical feedback at all stages and improved the article's text in terms of editorial quality and structure: Andreeva E

Availability of data and materials

Not applicable.

Financial support and sponsorship

This research was funded in part by the Austrian Science Fund (FWF) project SpyCoDe (No. 10.55776/F85).

Conflicts of interest

All authors declared that there are no conflicts of interest.

Ethical approval and consent to participate

Not applicable.

Consent for publication

This manuscript is based on the previous conference paper "A Forkcipher-Based Pseudo-Random Number Generator"[5]. All authors have approved and agreed on its submission to the journal.

Copyright

REFERENCES

1. Barak B, Halevi S. A model and architecture for pseudo-random generation with applications to /dev/random. In: Atluri V, Meadows C, Juels A, editors. CCS '05: Proceedings of the 12th ACM conference on Computer and communications security. New York, NY, USA: ACM Press; 2005. pp. 203-12. DOI
2. Dodis Y, Pointcheval D, Ruhault S, Vergnaud D, Wichs D. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In: Sadeghi AR, Gligor V, Yung M, editors. CCS '13: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. Berlin, Germany: ACM Press; 2013. pp. 647-58. DOI
3. Cohney S, Kwong A, Paz S, et al. Pseudorandom black swans: cache attacks on CTR_DRBG. In: 2020 IEEE Symposium on Security and Privacy (SP); 2020 May 18-21; San Francisco, CA, USA. IEEE; 2020. pp. 1241-58. DOI
4. Hoang VT, Shen Y. Security analysis of NIST CTR-DRBG. In: Micciancio D, Ristenpart T, editors. Advances in cryptology - CRYPTO 2020, Part I. Lecture notes in computer science. Cham: Springer; 2020. pp. 218-47. DOI
5. Andreeva E, Weninger A. A forkcipher-based pseudo-random number generator. In: Tibouchi M, Wang X, editors. Applied cryptography and network security. ACNS 2023. Lecture notes in computer science, vol 13906. Cham: Springer; 2023. pp. 3-31. DOI
6. Andreeva E, Lallemand V, Purnal A, Reyhanitabar R, Roy A, Vizár D. Forkcipher: a new primitive for authenticated encryption of very short messages. In: Galbraith SD, Moriai S, editors. Advances in cryptology - ASIACRYPT 2019. Lecture notes in computer science, vol 11922. Cham: Springer; 2019. pp. 153-82. DOI
7. Beierle C, Jean J, Kölbl S, et al. The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw M, Katz J, editors. Advances in cryptology - CRYPTO 2016, Part II. Lecture notes in computer science, vol. 9815. Heidelberg, Germany: Springer; 2016. pp. 123-53. DOI
8. Andreeva E, Cogliati B, Lallemand V, Minier M, Purnal A, Roy A. Masked iterate-fork-iterate: a new design paradigm for tweakable expanding pseudorandom function. 2022. Available from: https://eprint.iacr.org/2022/1534. [Last accessed on 22 Jan 2024]
9. Jean J, Nikolic I, Peyrin T, Seurin Y. The Deoxys AEAD family. *J Cryptol* 2021;34:31. DOI
10. Bhati AS, Dufka A, Andreeva E, Roy A, Preneel B. Skye: an expanding PRF based fast KDF and its applications. 2023. Available from: https://eprint.iacr.org/2023/781. [Last accessed on 22 Jan 2024]
11. Andreeva E, Bhati AS, Preneel B, Vizár D. 1, 2, 3, Fork: counter mode variants based on a generalized forkcipher. *IACR Trans Symmetric Cryptol* 2021;2021:1-35. DOI
12. Hoang VT, Tessaro S. Key-alternating ciphers and key-length extension: exact bounds and multi-user security. In: Robshaw M, Katz J, editors. Advances in cryptology - CRYPTO 2016. Lecture notes in computer science, vol. 9814. Berlin, Heidelberg: Springer; 2016. pp. 3-32. DOI
13. Chen S, Steinberger J. Tight security bounds for key-alternating ciphers. In: Nguyen PQ, Oswald E, editors. Advances in cryptology - EUROCRYPT 2014. Lecture Notes in Computer Science, vol. 8441. Berlin, Heidelberg: Springer; 2014. pp. 327-50. DOI
14. Patarin J. The "Coefficients H" technique. In: Avanzi RM, Keliher L, Sica F, editors. Selected areas in cryptography. Berlin, Heidelberg: Springer; 2009. pp. 328-45. DOI
15. Raz R, Reingold O. On recycling the randomness of states in space bounded computation. In: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing. Atlanta, GA, USA: ACM Press; 1999. pp. 159-68. DOI
16. Barak B, Dodis Y, Krawczyk H, et al. Leftover Hash Lemma, revisited. In: Rogaway P, editor. Advances in cryptology - CRYPTO 2011. Lecture notes in computer science, vol. 6841. Berlin, Heidelberg: Springer; 2011. pp. 1-20. DOI
17. Müller S. Butterknife;. Accessed: 2023-10-12. Available from: https://github.com/n0900/BC__Butterknife.
18. kokke. Tiny AES in C. https://github.com/kokke/tiny-AES-c. [Last accessed on 22 Jan 2024]
19. Pohle E. ForkSkinny-C. https://github.com/ErikP0/forkskinny-c. [Last accessed on 22 Jan 2024]