A Robust and Sponge-Like PRNG with Improved Efficiency

Daniel Hutchinson

Royal Holloway, University of London

Abstract. Ever since Keccak won the SHA3 competition, sponge-based constructions are being suggested for many different applications, including pseudo-random number generators (PRNGs). Sponges are very desirable, being well studied, increasingly efficient to implement and simplistic in their design. The initial construction of a sponge-based PRNG (Bertoni et al. CHES 2010) based its security on the well known sponge indifferentiability proof in the random permutation model and provided no forward security.

Since then, another improved sponge-based PRNG has been put forward by Gaži and Tessaro (Eurocrypt 2016) who point out the necessity for a public seed to prevent an adversarial sampler from gaining non-negligible advantage. The authors further update the security model of Dodis et al. (CCS 2013) to accommodate a public random permutation, modelled in the ideal cipher model, and how this affects the notions of security.

In this paper we introduce Reverie, an improved and practical, sponge-like pseudo-random number generator together with a formal security analysis in the PRNG with input security model of Dodis et al. with the modifications of the Gaži and Tessaro paper.

We prove that Reverie is *robust* when used with a public random permutation; robustness is the strongest notion of security in the chosen security model. Robustness is proved by establishing two weaker notions of security, preserving and recovering security, which together, can be shown to imply the robustness result. The proofs utilise the H-coefficient technique that has found recent popularity in this area; providing a very useful tool for proving the generator meets the necessary security notions.

Keywords: sponge; pseudo-random number generator (PRNG); Patarin's H-coefficient technique; robustness; Keccak; SHA-3; ideal permutation model

1 Introduction

Randomness is an essential ingredient in almost every area of cryptography; yet in the literature, randomness is often sampled uniformly at random with little thought on how quickly this amount of "good" randomness can be generated in practice. The need for high quality randomness delivered quickly has spawned

work on the various key aspects of a PRNG, such as the ability to produce randomness at a fast and reliable rate, and protection against adversaries who may be able to compromise parts of the generator's state or the environment in which it draws entropy. In practice, many generators in active use have not received valid security analysis, and, on the opposite side of the fence, many designs are created in a theoretical setting without the full scope of desirable properties for a PRNG in mind and as result, are impractical for active use.

Sponges The sponge design is very simple and yet very powerful; it benefits from a large amount of analysis due to the success of Keccak [6] in the SHA3 competition in 2012. The design requires an n-bit state with a rate r and capacity c such that $n = r + c$; the r bits of the state s are known as the *outer* state, written $\bar{s}$ while the c bits are known as the *inner* state $\widehat{s}$. The design initialises with an initial state of the zero state, and a random permutation π. The sponge has two algorithms; Absorb and Squeeze.

Previous constructions The sponge-based PRNG construction first suggested by Bertoni, Daemen, et al. in [8] utilises a random permutation and relies on the sponge indifferentiability proof of [7] for security. This analysis, though useful, does not consider security in terms of a security model for PRNGs. More recently, work by Gaži and Tessaro has improved upon this design and security claims, but still requires multiple additional calls to the permutation to ensure forward security, along with several additional strings to give a seeded design.

Ideal permutation model We prove all of our security claims in the ideal permutation model where π is a public, random permutation picked at the beginning of any game. Any party has access to the permutation and may make forward and backward queries. We denote by $\mathcal{A}^{\pi}$ an adversary with oracle access to $\pi \xleftarrow{\$} \mathcal{P}_n$ with $\mathcal{P}_n$ being the space of all permutations on n-bit strings. We say that $\mathcal{A}^{\pi}$ is a q_{π}-query adversary if it makes at most q_{π} queries π.

PRNG security models The development of security models for PRNGs has been slow due to a complex combination of security goals and the difficulty in accurately capturing the environment both the PRNG and the associated adversary are working in. Security models for PRNGs include work by Barak and Halevi [2], from 2005, a brilliantly simple model that introduced a very strong notion of Robustness.

This model was later improved upon in successive work by Dodis et al. [10], which initially aims to address the situation where a PRNG accumulates entropy at a slow rate, and is at risk of "prematurely" being called before enough entropy has been gathered.

The model was then further improved in [11], which introduced the idea of a scheduler, inspired by the design of the Fortuna PRNG [12] which aimed at a design to improve the recovery time of a compromised PRNG. We will not be considering a scheduler in this paper and will keep to the definitions of [10];

however, the idea of a scheduler is an interesting prospect in terms of possibly replacing the need for seed described below.

Seedless design More recently, work done concurrently to the first draft of this paper, by Gaži and Tessaro [13], concentrates on the importance of a "seeded" design when using a public ideal permutation. The authors argue that a publicly available permutation allows an adversary to generate PRNG inputs *dependent* on the permutation. These "bad" distributions can output high entropy inputs but result in a predictable bit of the state, and thus result in a non-negligible advantage for the adversary. The authors of this work ensure their implementation is seeded by requiring a small number ($s = 2$ or 3) of r-bit strings that are used as additional inputs to prevent this attack.

We note that this adds to the initial entropy requirements of the PRNG, which can already be one of the most restrictive and problematic situations for a PRNG. Another addition is the need for a counter to be kept; this is absorbed into the refresh procedure but should be an additional part of the state. Fortunately due to work in [15], this would not affect security. Alternatively, this could merely be an identifier of the system on which the PRNG is implemented, along with the current time of the system clock, which could be hashed to provide the seed, though in the security games the seed is chosen uniformly at random. Our design is aimed at being practical and efficient; in a practical scenario the distribution sampler or entropy accumulation mechanism is not so easily influenced and discovering these "bad" distributions is very difficult when good, studied entropy sources are used.

We include the option of a seed so that robustness can be achieved, but we question the necessity of the seed in a practical scenario; this can be likened to many PRNGs made for practical use having the option of a "personalisation" string [16], but note that this is often not used or even implemented. In practical implementations the PRNG does not have direct access to a noise source, but rather an entropy source that has been studied and provides a minimum entropy estimate, along with post processing and health checking [3,4].

Notation In this paper we denote by s_i the ith $n-$bit state of a generator. In the context of sponges we work with an $n-$bit state s_i which is split into an inner state of c-bits, denoted by $\widehat{s}_i$. The rest of the state is called the outer state, of r-bits and is denoted $\bar{s}_i$. Thus, the state can be given as $s_i = (\bar{s}_i \| \widehat{s}_i)$ where $\|$ is the usual concatenation of strings. The construction defined in this paper utilises a public, random permutation π from the set $\mathcal{P}_n$ of all permutations on n-bits. We use $x \xleftarrow{\$} X$ to denote an element x of a set X chosen uniformly at random. We denote by I_i the ith r-bit input string, used to refresh the state of a generator. We denote by r_i the ith output of a generator. These counters are in fact dependent on the state counter, so rather than the i-th output, we refer to the output associated with state i.

Contributions We put forward an improved sponge-like PRNG design which we prove is robust in the updated security model. The recent work by Gaži and

Tessaro updated the security of the sponge-based PRNG design of Bertoni et al. but did not seek to improve the design of the next procedure. We improve the design of the next function to ensure our design is more efficient, making a single call to the permutation π, compared with $1+t$ calls; resulting in a design better suited for practical application, especially those that restrict the number of calls to π. Since the $\mathsf{p.forget}$ procedure of the previous generator calls the permutation $1+t$ times, with zeroing, it presents the problem of increased collisions in the state, something that is avoided by our design and thus our bound is mainly limited by the collision factor associated with the refresh procedure. This potentially makes our generator comparatively more secure when first initialised on a random initial state and before any refreshes have been made. Below are the two main components of the new design Reverie.

- $\mathsf{Reverie.refresh}^{\pi}(s_i, \mathrm{I}, \mathsf{seed}, j) = \pi((\bar{s}_i \oplus \mathrm{I} \oplus \mathsf{seed}_j) \| \widehat{s}_i) = s_{i+1}, j = j+1 \mod s,$
- $\mathsf{Reverie.next}^{\pi}(s_i) = (\pi(s_i) \oplus (0^r \| \widehat{s}_i), \bar{s}_i) = (s_{i+1}, r_{i+1}).$

The security notion of interest in this paper is the strongest security notion, "robustness" which, informally, refers to an adversary working in time t, with access to a distribution sampler $\mathcal{D}$ that outputs refresh material used to update the state of the generator.

The adversary is allowed up to $q_{\mathcal{D}}$ outputs from the distribution sampler $\mathcal{D}$, these strings are required to have a minimum entropy when being used to refresh the generator from a compromised state. The adversary also has access to two algorithms $\mathsf{get\text{-}next}$ and $\mathsf{next\text{-}ror}$ which give the adversary output from the generator or random. The adversary is allowed up to q_R queries between these two algorithms.

Lastly, the adversary has up to q_S queries to $\mathsf{set\text{-}state}$ and $\mathsf{get\text{-}state}$ which give the adversary the current state of the generator and in the case of the former, allow it to set the state. In addition, the generator is said to be "uncompromised" if the current state has minimum entropy $\geq \gamma^*$ for some value γ^*. We say a generator is $((t, q_{\mathcal{D}}, q_R, q_S), \gamma^*, \epsilon)$ robust where ϵ is the maximum advantage of any adversary playing the robustness game.

The design can be seen in Figure 2 for further clarity. Although this design departs slightly from the sponge design, it can still be captured by the more generalised structure of the parazoa as defined in [1], and, given access to the underlying permutation function, easily implemented.

Organisation This paper is organised into preliminaries in Sections 2 and 3, followed by the description of the new generator in Section 4, the security analysis of the generator in Section 5 and finally a discussion of results in Section 6.

2 Preliminary Definitions

This section aims to provide a background on all the necessities of pseudo-random number generators (PRNG), the ideal permutation model, along with an introduction to Patarin's H-coefficient technique.

2.1 Probabilities and Further Notation

Definition 1. *The statistical distance between two discrete random variables X and Y over the set $\mathcal{X}$ is denoted*

$$\mathsf{SD}(X, Y) = \frac{1}{2} \sum_{x \in \mathcal{X}} \left| \Pr[X = x] - \Pr[Y = x] \right|.$$

Definition 2. *The minimum entropy of a random variable X is defined as* $H_\infty(X) = \mathsf{min}_{x \xleftarrow{\$} X}\{-\log(\Pr[X = x])\}$.

Definition 3. *For the purposes used in this paper, a source S^π is defined as an input-less randomised oracle which makes queries to π and outputs a string. The range of the source is denoted $[S]$ and is the set of all values the source outputs with positive probability, taken over the choice of π and the internal randomness of S.*

We use the usual game-based formalism from [5]; for a game G, $\mathsf{G}(\mathcal{A}) \Rightarrow 1$ denotes the event that an adversary $\mathcal{A}$ playing the game G, results in the game outputting 1, while $\mathsf{G}(\mathcal{A}) \rightarrow 1$ denotes the event that the $\mathcal{A}$ playing the game G outputs 1.

2.2 PRGs and PRNGs

In this document a PRG will refer to a pseudo-random number generator *without* input, while PRNG will refer to a pseudo-random number generator *with* input and in the form described in Definition 4.

Definition 4 (PRNG from [10]). *A PRNG with input is a triple of algorithms $G = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ and a triple $(n, \ell, p) \in \mathbb{N}^3$ where: n is the state length, ℓ is the output length, p is the input length of G and*

- setup: is a probabilistic algorithm that outputs some public parameters seed for the generator.
- $\mathsf{refresh}$: is a deterministic algorithm that, given seed, a state $s_i \in \{0,1\}^n$ and an input $\mathrm{I} \in \{0,1\}^p$, outputs a new state $s_{i+1} := \mathsf{refresh}(s_i, \mathrm{I}, \mathsf{seed})$
- next: is a deterministic algorithm that, given seed and a state $s_i \in \{0,1\}^n$, outputs a pair $(s_{i+1}, r_{i+1}) = \mathsf{next}(\mathsf{seed}, s_i)$, where s_{i+1} is the new state and $r_{i+1} \in \{0,1\}^\ell$ is the output. We write $\mathsf{next}(s_i)$ and omit seed for clarity.

Definition 5 (Originally of [10] but as amended in [13]). *A Q-distribution sampler is a randomised stateful oracle algorithm $\mathcal{D}$ which operates as follows:*

- It takes a state σ_i, with initial state $\sigma_0 = \perp$.
- $\mathcal{D}^{\pi}(\sigma_i)$ outputs a tuple $(\sigma_i, \mathcal{S}_i, \gamma_i, z_i)$, where
  - σ_i is the new state of $\mathcal{D}^{\pi}$.
  - $\mathcal{S}_i$ is a source with range $[\mathcal{S}_i] \subseteq \{0,1\}_i^{\ell}$ for some $\ell_i \geq 1$.
  - γ_i is an entropy estimation for $\mathcal{S}_i$ which will be discussed further below.
  - z_i is the leakage and/or auxiliary information about $\mathcal{S}_i$.
- When run $q_{\mathcal{D}}$ times, the number of queries to the permutation π made by $\mathcal{D}^{\pi}$ and $\mathcal{S}_1, \ldots, \mathcal{S}_{q_{\mathcal{D}}}$ is at most $Q(q_{\mathcal{D}})$.

For simplicity, $(\sigma_i, \mathrm{I}_i, \gamma_i, r_i) \xleftarrow{\$} \mathcal{D}^{\pi}(\sigma_{i-1})$ is written as the overall process of running $\mathcal{D}$ and the generated source $\mathcal{S}_i$. Next, we note the requirement for some restriction on distribution samplers, namely we require the following:

Definition 6. *A distribution sampler $\mathcal{D}$ as defined above in Definition 5 is $(q_{\mathcal{D}}, q_{\pi})$-legitimate, if, for every adversary $\mathcal{A}$ making q_{π} queries, every $i^* \in [q_{\mathcal{D}}]$, and for any possible values $(\mathrm{I}_j)_{j \neq i^*}, (\gamma_1, z_1), \ldots, (\gamma_{q_{\mathcal{D}}}, z_{q_{\mathcal{D}}}), V_{\mathcal{A}}, Q_{\mathcal{D}}$ potentially output by the game $\mathsf{GLEG}_{q_{\mathcal{D}}, i^*}(\mathcal{A}, \mathcal{D})$ with positive probability,*

$$\Pr\left[\mathrm{I}_{i^*} = x \mid (\mathrm{I}_j)_{j \neq i^*}, (\gamma_1, z_1), \ldots, (\gamma_{q_{\mathcal{D}}}, z_{q_{\mathcal{D}}}), V_{\mathcal{A}}, Q_{\mathcal{D}}\right] \leq 2^{-\gamma_{i^*}},$$

for all $x \in \{0,1\}^{\ell_{i^*}}$, where the probability is conditioned on these particular values being output by the game. The game $\mathsf{GLEG}_{q_{\mathcal{D}}, i^*}(\mathcal{A}, \mathcal{D})$, is defined in full in [13, Definition 3, page 10] and presented in Appendix A, but informally, the challenger samples a permutation π, $\mathcal{D}^{\pi}$ is run $q_{\mathcal{D}}$ times and the adversary $\mathcal{A}$ is run on all of the output from $\mathcal{D}^{\pi}$, apart from that of $\mathcal{S}_i$ and its associated queries. $V_{\mathcal{A}}$ is the adversary's final output, while $Q_{\mathcal{D}}$ is the input-output pairs of permutation queries made by $\mathcal{D}$.

2.3 The Ideal Permutation Model (IPM)

An implementation of a sponge-based PRNG would involve a publicly available permutation; hence, our analysis is done in the ideal permutation model. Formally, each party has oracle access to a public, random permutation $\pi \xleftarrow{\$} \mathcal{P}_n$, chosen by the challenger at the beginning of a game. The permutation can be queried as both π and π^{-1} but for simplicity, we write that an algorithm or entity, such as an adversary $\mathcal{A}$, has access to π by $\mathcal{A}^{\pi}$. We make use of the following, which denotes the advantage of an adversary $\mathcal{A}$ with oracle access to π in distinguishing between the distributions D_0, D_1 that also have access to π:

$$\mathsf{Adv}^{\mathsf{dist}}_{\mathcal{A}}(D_0, D_1) = \left|\Pr\left[X \xleftarrow{\$} D_0^{\pi} : \mathcal{A}^{\pi}(X) \Rightarrow 1\right] - \Pr\left[X \xleftarrow{\$} D_1^{\pi} : \mathcal{A}^{\pi}(X) \Rightarrow 1\right]\right|,$$

with $\mathcal{A}$ being called a q_π-query adversary if it asks at most q_π queries to π.

2.4 Patarin's H-Coefficient Technique

This section gives a brief introduction to Patarin's H-coefficient technique with a focus on functionality. Influenced by [9] and initially defined in [14], the H-coefficient technique is applied by splitting the "transcripts" of a game into two or more distinct sets; calculating the probability of the real or ideal world outputting transcripts in a particular set yields a close bound for the statistical distance of the real and ideal world.

A high level overview is that of a q-query information theoretic adversary $\mathcal{A}$ which can be assumed to be deterministic, making no redundant queries without loss of generality, interacting with an oracle ω representing either the real world or ideal world. The interaction $\mathcal{A}$ has with this oracle ω is represented in a transcript τ which includes a list of queries and their answers given by ω.

Let ω be an oracle that serves as the way the adversary $\mathcal{A}$ interacts with the challenger in the chosen world. Let Ω_X refer to the probability space of all real world oracles with the uniform probability distribution, and similarly Ω_Y is the probability space of all ideal world oracles again with the uniform distribution.

Let $\mathcal{T}$ be the set of all transcripts, with $\tau \in \mathcal{T}$ an individual transcript that describes, in full, the interactions and final output between the adversary $\mathcal{A}$ and the oracle she interacts with.

Further, the random variables X and Y are defined over the probability spaces respectively, where $X(\omega) = \tau$ refers to running $\mathcal{A}$ on oracle ω for $\omega \in \Omega_X$, which in turn produces the transcript τ.

For simplicity we will only consider two sets; good and bad transcripts, which are denoted $\mathcal{T}_G$ and $\mathcal{T}_B$ respectively. Defining this split is integral to the proof since the H-coefficient technique allows bounding the statistical distance of the

random variables X and Y in the following way: suppose $\exists \epsilon \in [0,1]$, such that $\forall \tau \in \mathcal{T}_G,$ with $\Pr[Y = \tau] > 0$,

$$\frac{\Pr[X = \tau]}{\Pr[Y = \tau]} \geq 1 - \epsilon.$$

Finally,

Theorem 1 (H-coefficient). *Let $X, Y, \mathcal{T}_G, \mathcal{T}_B, \tau, \epsilon$ be as above, then,*

$$\mathsf{SD}(X, Y) \leq \epsilon + \Pr[Y \in \mathcal{T}_B].$$

3 Security Notions

This section defines the notion of robustness originally from [10], but augmented as in [13] to allow for the publicly available random permutation. Robustness is the strongest security notion of the security model. We also include definitions of two weaker notions of security; preserving and recovering security, which together imply that a PRNG fulfils the requirements of robustness.

As per the definitions of [10], a minimal "fresh" entropy in the PRNG system when security should be expected. Minimising γ^* corresponds to a stronger security guarantee.

An adversary is modelled using a pair $(\mathcal{A}, \mathcal{D})$ where $\mathcal{A}$ is the actual q_π-query adversary and $\mathcal{D}$ is a $(q_\mathcal{D}, q_\pi)$-legitimate distribution sampler. The adversary $\mathcal{A}$'s goal is to determine a challenge bit b picked during the initialise procedure, this procedure also returns seed to the adversary.

Definition 7. *A PRNG with input G, is called $((q_\pi, q_\mathcal{D}, q_R, q_S), \gamma^*, \epsilon_{\mathsf{rob}})$-robust ($\mathsf{ROB}_G^{\gamma^*}$) if for any adversary $\mathcal{A}$ making at most q_π queries to $\pi^{\pm}$, making at most $q_\mathcal{D}$ calls to $\mathcal{D}$-refresh, q_R calls to Next-ror/Get-next and q_S calls to Get-state/Set-state and any legitimate distribution sampler $\mathcal{D}$, the advantage of any adversary in the robustness game is at most ϵ_{rob} which is defined below.*

The adversary $\mathcal{A}$ has access to a subset of the following oracles, dependent on the security game that it's playing; the full set is available in $\mathsf{ROB}_G^{\gamma^*}(\mathcal{A}, \mathcal{D})$. We say that an adversarial pair $(\mathcal{A}, \mathcal{D})$ playing the robustness game as described below in Section 3 for a PRNG G have advantage

$$\mathsf{Adv}_{\mathsf{G}}^{\gamma^* - \mathsf{ROB}}(\mathcal{A}, \mathcal{D}) := \left| 2\Pr\left[\mathsf{ROB}_G^{\gamma^*}(\mathcal{A}, \mathcal{D}) \Rightarrow 1\right] - 1 \right| \leq \epsilon_{\mathsf{rob}}.$$

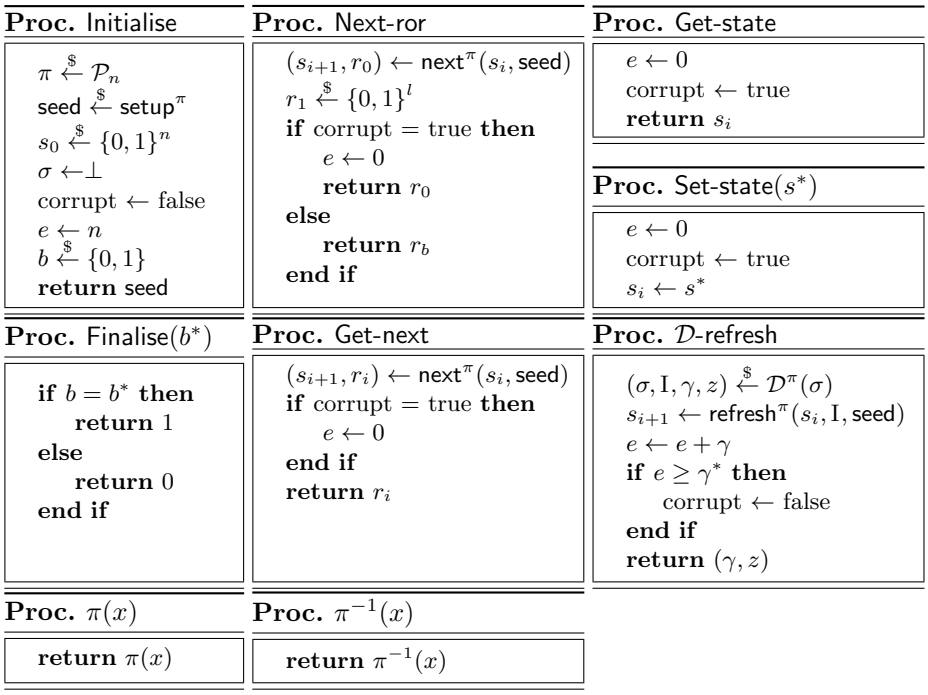

Fig. 1: $\mathsf{ROB}_G^{\gamma^*}(\mathcal{A}, \mathcal{D})$

Next, we define two further security notions: preserving security and recovering security. If a PRNG satisfies both these notions, then by Theorem 1 of [10] (with updated version from [13]) the generator in question satisfies the robustness security notion under the corresponding parameters. Next we define preserving and recovering security.

3.1 Preserving Security

Informally, preserving security states that if the state of a generator starts uncompromised, is refreshed using compromised input, then the next output and resulting state are still indistinguishable from random.

Definition 8. *A PRNG with input is said to have $(q_\pi, \epsilon_{\mathsf{pres}})$-preserving security if the advantage of any adversary $\mathcal{A}$ making at most q_π queries to $\pi^{\pm}$ in the following game is at most ϵ_{pres}, where the advantage is defined to be*

$$\mathsf{Adv}_{\mathsf{G}}^{\mathsf{PRES}}(\mathcal{A}) := |2\Pr\left[\mathsf{PRES}_G(\mathcal{A}) \Rightarrow 1\right] - 1| \leq \epsilon_{\mathsf{pres}}.$$

$\mathsf{PRES}_G(\mathcal{A})$

$\pi \xleftarrow{\$} \mathcal{P}_n, \mathsf{seed} \xleftarrow{\$} \mathsf{setup}^\pi(), b \xleftarrow{\$} \{0,1\}, s_0 \xleftarrow{\$} \{0,1\}^n$

$(\mathrm{I}_1, \ldots, \mathrm{I}_d) \leftarrow \mathcal{A}^\pi(\mathsf{seed})$

for $j = 1, \ldots, d$ **do**

  $s_j \leftarrow \mathsf{refresh}^\pi(s_{j-1}, \mathrm{I}_j, \mathsf{seed})$ **if** $b = 0$ **then** $(S, T) \leftarrow \mathsf{next}^\pi(s_d, \mathsf{seed})$

else $(S, T) \xleftarrow{\$} \{0,1\}^n \times \{0,1\}^r$

$b^* \leftarrow \mathcal{A}^\pi((S, T))$

return $b == b^*$

3.2 Recovering Security

Informally, recovering security implies that if a PRNG is compromised, inserting enough random entropy to refresh the internal state will ensure that the next output and state will be indistinguishable from random.

Definition 9. *A PRNG with input has $(q_\pi, q_\mathcal{D}, \gamma^*, \epsilon_{\mathsf{rec}})$-recovering security if the advantage of any adversary $\mathcal{A}$ making at most $q_\mathcal{D}$ queries to $\pi^\pm$ and distribution sampler $\mathcal{D}$, making at most $Q(q_\mathcal{D})$ queries to $\pi^\pm$, in the following game with $\gamma^* > 0$ is at most ϵ_{rec} where advantage is defined as*

$$\mathsf{Adv}_{\mathsf{G}}^{(\gamma^*, \mathsf{q}_\mathcal{D})-\mathsf{rec}}(\mathcal{A}, \mathcal{D}) := \left| 2\Pr\left[\mathsf{REC}_G^{(\gamma^*, q_\mathcal{D})} \Rightarrow 1\right] - 1 \right| \leq \epsilon_{\mathsf{rec}}.$$

$\mathsf{REC}_G^{(\gamma^*, q_\pi)}(\mathcal{A}, \mathcal{D})$

$\pi \xleftarrow{\$} \mathcal{P}_n, \mathsf{seed} \xleftarrow{\$} \mathsf{setup}^\pi(), b \xleftarrow{\$} \{0,1\}, \sigma_0 \leftarrow \perp$

for $k = 1, \ldots, q_\mathcal{D}$ **do**

  $(\sigma_k, \mathrm{I}_k, \gamma_k, z_k) \leftarrow \mathcal{D}^\pi(\sigma_{k-1})$

$k \leftarrow 0$

$(s_0, d) \leftarrow \mathcal{A}^{\pi, \mathsf{get\text{-}refresh}()}(\gamma_1, \ldots, \gamma_{q_\mathcal{D}}, z_1, \ldots, z_{q_\mathcal{D}}, \mathsf{seed})$

if $k + d > q_\mathcal{D}$ **then return** $\perp$

else

if $\sum_{j=k+1}^{k+d} \gamma_j < \gamma^*$ **then return** $\perp$

else

for $j = 1, \ldots, d$ **do**

  $s_j \leftarrow \mathsf{refresh}^\pi(s_{j-1}, \mathrm{I}_{k+j}, \mathsf{seed})$

if $b = 0$ **then** $(S, T) \leftarrow \mathsf{next}^\pi(\mathsf{seed}, s_d)$

else $(S, T) \xleftarrow{\$} \{0,1\}^n \times \{0,1\}^r$

$b^* \leftarrow \mathcal{A}^\pi((S, T), \mathrm{I}_{k+d+1}, \ldots, \mathrm{I}_{q_\mathcal{D}})$

return $b == b^*$

Oracle $\mathsf{get\text{-}refresh}()$

$k \leftarrow k + 1$

return I_k

4 Improved Construction

The following algorithms describe Reverie, a sponge-like PRNG with forward security that does not require additional calls to the underlying public permutation. Let $s, r, c \geq 1$ and $c := n - r, \ell = p = r$, together with $\pi \xleftarrow{\$} \mathcal{P}_n$, then $\mathsf{Rev}^{\pi}_{s,n,r} := (\mathsf{Reverie.setup}^{\pi}, \mathsf{Reverie.refresh}^{\pi}, \mathsf{Reverie.next}^{\pi})$ for:

Proc. $.\mathsf{setup}^{\pi}()$	**Proc.** $.\mathsf{refresh}^{\pi}(s_i, \mathrm{I}, \mathsf{seed})$	**Proc.** $.\mathsf{next}^{\pi}(s_i, \mathsf{seed})$

for $i = 0, \ldots, s-1$ **do**
$\quad \mathsf{seed}_i \xleftarrow{\$} \{0,1\}^r$
end for
$\mathsf{seed} \leftarrow$
$(\mathsf{seed}_0, \ldots, \mathsf{seed}_{s-1})$
$j \leftarrow 1$
	$s_{i+1} \leftarrow \pi((\bar{s}_i \oplus \mathrm{I} \oplus \mathsf{seed}_j) \| \widehat{s}_i)$
	$j \leftarrow j + 1 \mod s$
	return s_{i+1}
		$r_{i+1} \leftarrow \bar{s}_i$
		$s_{i+1} \leftarrow$
		$(\pi(s_i) \oplus (0^r \| \widehat{s}_i))$
		return (s_{i+1}, r_{i+1})

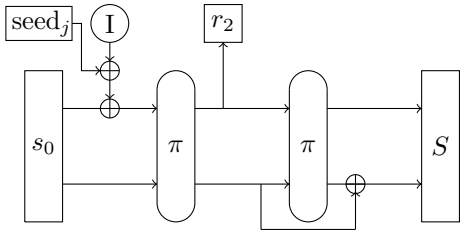

Fig. 2: Reverie.

5 Security of Reverie

This section consists of the security proofs of Reverie; the approach is to analyse the security of the next function, and then focus on the preserving and recovering security games, making use of the previous result.

Theorem 2. *For Reverie* $= \mathsf{Rev}^{\pi}_{s,n,r}$ *as defined above, let* $\gamma^* > 0$*, let* $\mathcal{D}$ *be a* $(q_{\mathcal{D}}, q_{\pi})$*-legitimate distribution sampler, let* $\bar{q}_{\pi} := q_{\pi} + Q(q_{\mathcal{D}})$ *and* $\widehat{q} := \bar{q}_{\pi} + q_R + q_{\mathcal{D}} d$*. Then* $\mathsf{Rev}^{\pi}_{s,n,r}$ *is* $((q_{\pi}, q_{\mathcal{D}}, q_R, q_S), \gamma^*, \epsilon_{\mathsf{rob}})$*-robust, for* ϵ_{rob} *as below:*

$$\mathsf{Adv}^{\gamma^*-\mathsf{rob}}_{\mathsf{Rev}^{\pi}_{s,n,r}}(\mathcal{A}, \mathcal{D}) \leq q_R \cdot \left(\frac{\bar{q}_{\pi} + 1}{2^{\gamma^*}} + \frac{Q(q_{\mathcal{D}})}{2^{sr}} + \frac{7(\widehat{q}^2 + 1) + 29\widehat{q}}{2^{c-1}} + \frac{(2d^2 + 3)\widehat{q} + d(3d + 2d)}{2^n} \right).$$

Proof. The theorem is the result of the preserving and recovering security bounds in Lemmas 2 and 3 respectively, combined by [13, Theorem 4].

Lemma 1 (Security of the next function). *Let U_x is the uniform distribution over x-bit strings, let* next *be as defined in Section 4, let $s_0 \xleftarrow{\$} \{0,1\}^n$, then for any q_π-query adversary $\mathcal{A}$,*

$$\begin{aligned}\epsilon_{\mathsf{next}}(q_\pi) := \mathsf{Adv}^{\mathsf{dist}}_{\mathcal{A}}(\mathit{next}(U_n), (U_n, U_r)) &\leq \left(2 - \frac{1}{2^r}\right)\frac{q_\pi}{2^{c-1}} + \frac{3q_\pi}{2^{c-1}} \\ &= \left(5 - \frac{1}{2^r}\right)\frac{q_\pi}{2^{c-1}}.\end{aligned}$$

Proof outline Distinguishing between $\mathsf{next}(s_0)$ and random output $(S,T) \xleftarrow{\$} \{0,1\}^n \times \{0,1\}^r$ naively, it seems like the adversary's only option is to guess the inner state of the secret initial state, by either a direct forward query to π or by an indirect guess that would reveal a candidate for this inner state through a query to π^{-1}.

The proof, given in Appendix B proves that this is in fact the optimal strategy. Since there are two parts to the challenge, the logical approach is to split the proof into first proving that one part of the challenge can be replaced with random, before approaching the remaining part of the challenge.

We note that unlike [13], the next function requires a uniformly random state; the difference is made up for in a game jump in the proof, but allows us to avoid an additional call to π, as is required in [13]. This step can be reinstated at the cost of a single additional call to π.

5.1 Preserving Security

Now that we have this tool, we can prove the following:

Lemma 2. *Given* Reverie *as defined in Section 4, and with ϵ_{next} as above, then for every q_π-query adversary $\mathcal{A}$ playing the preserving security game defined in Definition 8 with d adversarial refresh inputs, we have*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{pres}}_{\mathcal{A}}(\mathit{Reverie}[\pi]) &\leq \epsilon_{\mathsf{next}}(q_\pi) + \frac{q'_\pi + d}{2^n} + \frac{(d+1)(2q'_\pi + d)}{2^n} \\ &\leq \frac{5q_\pi}{2^{c-1}} + \frac{(2d+3)q_\pi + d(d+2)}{2^n}.\end{aligned}$$

Proof outline The proof relies on proving that for a random secret initial state s_0, the resulting state s_d will look random and thus, by our previous analysis of the next function, the challenge output will also be random. The full proof is located in Appendix C.

5.2 Recovering Security

Thanks to the impressive result of [13] the proof of recovering security can be expressed as an adaptation of their result; using the sponge as an extractor, and the security of the next function. To formalise this:

Lemma 3. *Let $q_\pi, \bar{q}_\pi := q_\pi + Q(q_\mathcal{D}), r, s, c$ be as in Section 4. Let $\epsilon_{\mathsf{ext}}(q_\pi, q_\mathcal{D})$ be as described in [13, Section 5.3] and similarly let $\epsilon_{\mathsf{next}}(\bar{q}_\pi)$ be the bound as in Lemma 1 as a function of $\bar{q}_\pi$; both with n, r, c as previously described. Given* Reverie*, also as in Section 4, $\gamma^* > 0, q_\mathcal{D} \geq 0, \mathcal{A}$, a q_π-query adversary against recovering security, and $\mathcal{D}$, a $(q_\mathcal{D}, q_\pi)$-legitimate distribution sampler as defined in Definition 5. Then,*

$$\begin{aligned}
&\mathsf{Adv}^{(\gamma^*,\mathsf{q}_\pi)-\mathsf{rec}}_{\mathsf{Rev}^\pi_{\mathsf{s,n,r}}}(\mathcal{A},\mathcal{D}) \leq \epsilon_{\mathsf{ext}}(q_\pi+1, q_\mathcal{D}) + 2\epsilon_{\mathsf{next}}(\bar{q}_\pi) + \frac{q_\pi}{2^{n-1}} \\
&\leq \frac{\bar{q}_\pi+1}{2^{\gamma^*}} + \frac{Q(q_\mathcal{D})}{2^{sr}} + \frac{7(\bar{q}_\pi^2+1)+24\bar{q}_\pi}{2^{c-1}} + \frac{(\bar{q}_\pi+1)d + d^2 + q_\pi - 2\bar{q}_\pi}{2^{n-1}}.
\end{aligned}$$

Proof outline The strategy of the proof is to use the extractor properties of the sponge to replace the resulting state with a random state; following this the output of next will be random by the arguments of Lemma 1. The full proof is located in Appendix D.

6 Conclusion

We have presented an updated construction, Reverie, for a sponge-like PRNG. The construction incorporates an effective and efficient forward-security mechanism and we have provided proofs of both preserving and recovering security in the chosen security model. Our design makes a single call to the permutation on every invocation of Reverie.next, while the comparable generators make $1 + t$ calls. Our design choice ensures the underlying permutation is called far fewer times. Thus, the loss of security from collisions is reduced when compared to the relevant bounds of other designs.

The main limiting factor of the bound relates to the recovering security bound; and more precisely the extraction bound. This begs the question: can this bound be improved? This is briefly discussed in [13] in the present setting, but we would also like to consider other, possibly similar mechanisms that may present a better security bound; for instance, would a full state refresh yield a better bound? A full state refresh however, enables in practise an adversary to more easily affect or even set the state of the generator.

References

[1] E. Andreeva, B. Mennink, and B. Preneel. The parazoa family: Generalizing the sponge hash functions. Cryptology ePrint Archive, Report 2011/028, 2011. http://eprint.iacr.org/2011/028.

[2] B. Barak and S. Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM CCS 05*, pages 203–212, Alexandria, Virginia, USA, Nov. 7–11, 2005. ACM Press.

[3] E. Barker and J. Kelsey. Recommendation for random number generation using deterministic random bit generators, sp800-90a. http://csrc.nist.gov/publications/drafts/800-90/sp800_90c_second_draft.pdf, April 2016.

[4] E. Barker and J. Kelsey. Recommendation for the entropy sources used for random bit generation. http://csrc.nist.gov/publications/drafts/800-90/sp800-90b_second_draft.pdf, Jan 2016.

[5] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[6] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The Keccak reference, January 2011. http://keccak.noekeon.org/.

[7] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197, Istanbul, Turkey, Apr. 13–17, 2008. Springer, Heidelberg, Germany.

[8] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge-based pseudo-random number generators. In S. Mangard and F.-X. Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 33–47, Santa Barbara, California, USA, Aug. 17–20, 2010. Springer, Heidelberg, Germany.

[9] S. Chen and J. P. Steinberger. Tight security bounds for key-alternating ciphers. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 327–350, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[10] Y. Dodis, D. Pointcheval, S. Ruhault, D. Vergnaud, and D. Wichs. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 647–658, Berlin, Germany, Nov. 4–8, 2013. ACM Press.

[11] Y. Dodis, A. Shamir, N. Stephens-Davidowitz, and D. Wichs. How to eat your entropy and have it too - optimal recovery strategies for compromised RNGs. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 37–54, Santa Barbara, CA, USA, Aug. 17–21, 2014. Springer, Heidelberg, Germany.

[12] N. Ferguson and B. Schneier. *Practical cryptography*. Wiley, 2003.

[13] P. Gazi and S. Tessaro. Provably robust sponge-based prngs and kdfs. In M. Fischlin and J. Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 87–116. Springer, 2016.

[14] J. Patarin. The "coefficients h" technique. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 2008.

[15] T. Shrimpton and R. S. Terashima. A provable-security analysis of Intel's secure key RNG. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 77–100, Sofia, Bulgaria, Apr. 26–30, 2015. Springer, Heidelberg, Germany.

[16] M. S. Turan, E. Barker, J. Kelsey, K. A. McKay, M. L. Baish, and M. Boyle. Recommendation for random number generation using deterministic random bit generators, sp800-90a. `http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf`, June 2015.

A $\mathsf{GLEG}_{q_{\mathcal{D}},i^*}(\mathcal{A},\mathcal{D})$

Below the full game $\mathsf{GLEG}_{q_{\mathcal{D}},i^*}(\mathcal{A},\mathcal{D})$ is given, as in [13, Definition 3, page 10] and following on from Definition 6:

Let $\mathcal{D}$ be a distribution sampler, $\mathcal{A}$ an adversary and fix an $i^* \in [q_{\mathcal{D}}]$. Let $Q_{\mathcal{D}}$ be the set of all input-output pairs of permutation queries made by $\mathcal{D}$ and by all S_j for $j \in [q_{\mathcal{D}}]/\{i^*\}$.

Game $\mathsf{GLEG}_{q_\pi,i^*}(\mathcal{A},\mathcal{D})$

$\pi \xleftarrow{\$} \mathcal{P}_n$
for $j = 1,\ldots,q_{\mathcal{D}}$ **do**
  $(\sigma_i, S_i, \gamma_i, z_i) \xleftarrow{\$} \mathcal{D}^{\pi}$
  $\mathrm{I}_i \xleftarrow{\$} S_i^{\pi}$
endfor
$V_{\mathcal{A}} \xleftarrow{\$} \mathcal{A}((\gamma_j, z_j)_{j\in[q_{\mathcal{D}}]}, (I_j)_{j\in[q_{\mathcal{D}}]/i^*})$
return $((\mathrm{I}_1,\gamma_1,z_1),\ldots,(\mathrm{I}_{q_{\mathcal{D}}},\gamma_{q_{\mathcal{D}}},z_{q_{\mathcal{D}}}),V_{\mathcal{A}},Q_{\mathcal{D}})$

Then $\mathcal{D}$ is said to be a $(q_{\mathcal{D}},q_\pi)$-legitimate distribution sampler if for every adversary $\mathcal{A}$ making q_π queries and every $i^* \in [q_{\mathcal{D}}]$, all possible values of $(\mathrm{I}_j)_{j\in[q_{\mathcal{D}}]/(i^*)}, (\gamma_1,z_1),\ldots,(\gamma_{q_{\mathcal{D}}},z_{q_{\mathcal{D}}}), V_{\mathcal{A}}, Q_{\mathcal{D}}$ potentially output by the above game with positive probability,

$$\Pr\left[\mathrm{I}_{i^*} = x \mid (\mathrm{I}_j)_{j\neq i^*}, (\gamma_1,z_1),\ldots,(\gamma_{q_{\mathcal{D}}},z_{q_{\mathcal{D}}}), V_{\mathcal{A}}, Q_{\mathcal{D}}\right] \leq 2^{-\gamma_{i^*}},$$

for all $x \in \{0,1\}$.

B Proof of next Security

Proof. Lemma 1

Algorithm 1 $\mathsf{next}_0^{\pi}(s_0)$

$s_0 \xleftarrow{\$} \{0,1\}^n$
$T \leftarrow \bar{s}$
$t \leftarrow \pi(s_0)$
$S \leftarrow t \oplus (0^r \| \widehat{s}_0)$

Algorithm 2 $\mathsf{next}_1^{\pi}(s_0)$

$s_0 \xleftarrow{\$} \{0,1\}^n$
$T \xleftarrow{\$} \{0,1\}^r$
$t \leftarrow \pi(s_0)$
$S \leftarrow t \oplus (0^r \| \widehat{s}_0)$

Algorithm 3 $\mathsf{next}_2(s_0)$

$s_0 \xleftarrow{\$} \{0,1\}^n$
$T \xleftarrow{\$} \{0,1\}^r$
$S \xleftarrow{\$} \{0,1\}^n$

These algorithms are set up so that on input $s_0 \xleftarrow{\$} \{0,1\}^n$, next_0 is precisely the next function on input s_0 while next_2 has the same distribution as (U_n, U_r). next_1^{π} will be used as a hybrid game. Thus, by the triangle inequality,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{dist}}_{\mathcal{A}}(\mathsf{next}(s_0),(U_n,U_r)) \leq & \mathsf{Adv}^{\mathsf{dist}}_{\mathcal{A}}(\mathsf{next}^{\pi}_0(s_0),\mathsf{next}^{\pi}_1(s_0)) \\ & + \mathsf{Adv}^{\mathsf{dist}}_{\mathcal{A}}(\mathsf{next}^{\pi}_1(s_0),\mathsf{next}_2(s_0)).\end{aligned}$$

What follows is to prove the bound using the H-coefficient technique. As described in Section 2.4, we assume that $\mathcal{A}$ is deterministic and makes q_π non-repeating queries to the permutation π, denoted as

$$\tau_{\mathcal{A}} := (x_1, y_1, z_1), \ldots, (x_{q_\pi}, y_{q_\pi}, z_{q_\pi})$$

where $\forall i \in [1, \ldots, q_\pi]$,

$$\begin{aligned} y_i &= \pi(x_i), \\ z_i &= y_i \oplus (0^r \| \widehat{x}_i). \end{aligned}$$

In addition to the challenge, the adversary in this distinguishing game is also given several other pieces of information at the end of the game, after all queries to π have been made, but before the adversary must output her decision. Formally, $\mathcal{A}$ is given $\widehat{s}_0$ and $t' := (\overline{S} \| (\widehat{s}_0 \oplus \widehat{S}))$ which it can compute for itself but is given for clarity. This completes the definition of a transcript for these experiments,

$$\tau := ((x_1, y_1, z_1), \ldots, (x_{q_\pi}, y_{q_\pi}, z_{q_\pi}), \widehat{s}_0, t', (S, T)).$$

We say a transcript τ is compatible with $\mathsf{next}^{\pi}_0(s_0)$ if it can be output in the experiment where $\mathcal{A}$ receives $\mathsf{next}^{\pi}_0(s_0)$. Since $\mathsf{next}^{\pi}_1(s_0)$ and $\mathsf{next}_2(s_0)$ differ only by replacing real output with random, it's clear that if a transcript is compatible with $\mathsf{next}^{\pi}_0(s_0)$ then it is compatible with $\mathsf{next}^{\pi}_1(s_0)$ and $\mathsf{next}_2(s_0)$.

What follows is bounding the probability of different transcripts from each experiment.

Lemma 4. *For the experiments $\mathsf{next}^{\pi}_0(s_0), \mathsf{next}^{\pi}_1(s_0)$ as described above,*

$$\mathsf{Adv}^{\mathsf{dist}}_{\mathcal{A}}(\mathsf{next}^{\pi}_0(s_0), \mathsf{next}^{\pi}_1(s_0)) \leq \left(2 - \frac{1}{2^r}\right) \frac{q_\pi}{2^{c-1}} + 0 = \left(2 - \frac{1}{2^r}\right) \frac{q_\pi}{2^{c-1}}.$$

Proof. First we define the bad transcripts for this pair of experiments:

Definition 10 (Bad transcripts $\mathcal{T}_B$ for ($\mathsf{next}^{\pi}_0(s_0)\mathsf{next}^{\pi}_1(s_0)$)). *A compatible transcript as above, is called a bad transcript if any of the following occur:*

State Collision (SC): $\exists j \in [q_\pi]$ such that $x_j = (T \| \widehat{s}_0)$,
Image Collision (IC): $\exists j \in [q_\pi]$ such that $y_j = t'$,

The set of bad transcripts is denoted $\mathcal{T}_B$.

Let X_0, Y_0 be the random variables outputting transcripts that describe when $\mathcal{A}$ interacts with $\mathsf{next}_0^\pi(s_0)$ and $\mathsf{next}_1^\pi(s_0)$ respectively.

Lemma 5. *For an adversary making no more than $q_\pi \leq 2^{c-1}$ queries to an oracle in the experiment $\mathsf{next}_1(s_0)$,*

$$\Pr[Y_0 \in \mathcal{T}_B] \leq \left(2 - \frac{1}{2^r}\right) \frac{q_\pi}{2^{c-1}}.$$

Proof. Note that if $Y_0 \in \mathcal{T}_B$ then $\mathrm{SC} \vee \mathrm{IC}$ must occur.

$$\Pr[Y_0 \in \mathcal{T}_B] \leq \Pr[\mathrm{SC}] + \Pr[\mathrm{IC} \mid \neg\mathrm{SC}],$$

The first probability is relatively easy to bound,

$$\Pr[\mathrm{SC}] \leq \frac{q_\pi}{2^{c-1}}. \tag{B.1}$$

Since the adversary is given T at the start of the game and s_0 is uniformly distributed over all the 2^c n-bit strings with outer bits equal to T, and recalling that $q_\pi \leq 2^{c-1}$, the probability that $\mathcal{A}$'s i-th query is of the form $((T\|\widehat{s}_0), y_i, z_i)$ is $\frac{1}{2^c - i + 1}$. More formally, let $\Pr[win_i] := \Pr[x_i = (T\|\widehat{s}_0)]$, then

$$\begin{aligned}\Pr[win] &\leq \sum_{i=1}^{q_\pi} \Pr[win_i] = \sum_{i=1}^{q_\pi} \frac{1}{2^c - i + 1} \\ &\leq \sum_{i=1}^{q_\pi} \frac{1}{2^c - 2^{c-1}} = \frac{q_\pi}{2^{c-1}}.\end{aligned}$$

The second, since SC has not occurred, must be where the adversary is interacting with $\mathsf{next}_1^\pi(s_0)$ where T was chosen uniformly at random from r-bit strings, and as such, was not used to produce S. There is the situation that the randomly chosen T matches the real value of $\bar{s}_0$ which is reflected in the factor of $\left(1 - \frac{1}{2^r}\right)$.

The second probability is similar, in that the adversary has knowledge of $\overline{S}$, with $(\overline{S}\|(\widehat{s}_0 \oplus \widehat{S}))$ uniformly distributed over all the 2^c n-bit strings with outer bits equal to $\overline{S}$. It is also assumed that a SC has not occurred, meaning nothing beyond $\widehat{s}_0$ is known about s_0, then similarly to above,

$$\Pr[\mathrm{IC} \mid \neg\mathrm{SC}] \leq \left(1 - \frac{1}{2^r}\right) \frac{q_\pi}{2^{c-1}}. \tag{B.2}$$

Equation (B.2), together with Equation (B.1) complete the lemma.

Lemma 6. *For all compatible transcripts $\tau \in \mathcal{T}_G$,*

$$\Pr[X_0 = \tau] = \Pr[Y_0 = \tau].$$

Proof. For all $\tau \in \mathcal{T}_G$ (and for $\pi \xleftarrow{\$} \mathcal{P}_n$),

$$\begin{aligned}\Pr[X_0 = \tau] = \\ &\Pr[\forall i \in [q_\pi], \pi(x_i) = y_i] \cdot \Pr\left[\pi(s_0) = \left(\overline{S} \| (\widehat{s}_0 \oplus \widehat{S})\right) \mid \neg\mathrm{SC} \vee \neg\mathrm{IC}\right] \\ &= \frac{1}{2^r} 2^r \frac{(2^n - q_\pi - 1)!}{2^n!} = \Pr[Y_1 = \tau].\end{aligned}$$

Putting Lemmas 5 and 6 together yields the result.

Next, we prove the following:

Lemma 7. *For the experiments $\mathsf{next}_1^\pi(s_0), \mathsf{next}_2(s_0)$ as described above and by Theorem 1,*

$$\mathsf{Adv}^{dist}_{\mathcal{A}}(\mathsf{next}_1^\pi(s_0), \mathsf{next}_2(s_0)) \leq \frac{3q_\pi}{2^{c-1}} + 0 = \frac{3q_\pi}{2^{c-1}}.$$

Proof. This time, the transcript is slightly different, in that the adversary is given the entire s_0 at the end of her queries to π, so

$$\tau := ((x_1, y_1, z_1), \ldots, (x_{q_\pi}, y_{q_\pi}, z_{q_\pi}), s_0, t', (S, T)).$$

Comparing the distributions of these two experiments yields one more bad event, along with a modified state collision and unchanged image collision:

Definition 11 (Bad transcripts $\mathcal{T}_B$ for $(\mathsf{next}_1^\pi(s_0), \mathsf{next}_2(s_0))$). *A compatible transcript as above, is called a bad transcript if any of the following occur:*

$$\begin{aligned}\text{State Collision (SC): } &\exists j \in [q_\pi] \text{ such that } x_j = s_0, \\ \text{Image Collision (IC): } &\exists j \in [q_\pi] \text{ such that } y_j = t', \\ \text{Inversion (IN): } &\exists j \in [q_\pi] \text{ such that } z_j = S.\end{aligned}$$

The set of bad transcripts is denoted $\mathcal{T}_B$.

Let X_1, Y_1 be the random variables outputting transcripts that describe when $\mathcal{A}$ interacts with $\mathsf{next}_1^\pi(s_0)$ and $\mathsf{next}_2(s_0)$ respectively.

Lemma 8. *For an adversary making no more than $q_\pi \leq 2^{c-1}$ queries to an oracle in the experiment $\mathsf{next}_2(s_0)$,*

$$\Pr[Y_1 \in \mathcal{T}_B] \leq \frac{q_\pi}{2^{n-1}} + \left(2 - \frac{1}{2^r}\right) \frac{q_\pi}{2^{c-1}} = \frac{2q_\pi}{2^{c-1}}.$$

Proof. Note that if $Y_1 \in \mathcal{T}_B$ then SC $\vee$ IC $\vee$ IN must occur.

$$\Pr[Y_1 \in \mathcal{T}_B] \leq \Pr[\text{SC}] + \Pr[\text{IC} \mid \text{SC}] + \Pr[\text{IN} \mid \neg\text{SC} \wedge \neg\text{IC}],$$

The first probability is similar to before, but this time the adversary knows that $\overline{S}$ (with high probability) was not queried to π to produce the challenge. This results in the following:

$$\Pr[\text{SC}] \leq \frac{q_\pi}{2^{n-1}}.$$

The second probability is similar to the case where an IC occurs in a transcript in either $\mathsf{next}_0^p(s_0)$ or $\mathsf{next}_1^\pi(s_0)$. Once again since $\widehat{s}_0$ is uniformly distributed over $\{0,1\}^c$, the probability that any of the adversary's queries $(x_i) = y_i$ or $\pi^{-1}(y_i) = x_i$ is such that $y_i = (\overline{S} \| \widehat{S} \oplus \widehat{s}_0)$ is at most $\frac{1}{2^c - i + 1}$ resulting in the bound $\frac{q_\pi}{2^{c-1}}$. It is also assumed that a SC has not occurred, meaning nothing beyond $\widehat{s}_0$ is known about s_0. Thus,

$$\Pr[\text{IC} \mid \neg\text{SC}] \leq \left(1 - \frac{1}{2^r}\right) \frac{q_\pi}{2^{c-1}}.$$

Lastly, if neither a SC or IC has occurred, the probability of an IN can be expressed as

$$\Pr\left[\pi^{-1}(\overline{S} \| \widehat{y}_i) = \left(\widehat{x}_i \| (\widehat{y}_i \oplus \widehat{S})\right)\right],$$

which again is bounded by $\frac{q_\pi}{2^{c-1}}$ and together with the other events, yields the desired bound.

Lemma 9. *For all compatible transcripts $\tau \in \mathcal{T}_G$,*

$$\Pr[X_1 = \tau] = \Pr[Y_1 = \tau].$$

For all $\tau \in \mathcal{T}_G$ (and for $\pi \xleftarrow{\$} \mathcal{P}_n$),

Proof.

$$\begin{aligned}\Pr[X_1 = \tau] &= \Pr[\forall i \in [q_\pi], \pi(x_i) = y_i] \cdot \Pr\left[\pi(s_0) = \left(\overline{S} \| (\widehat{s}_0 \oplus \widehat{S})\right) \mid \neg\text{SC} \vee \neg\text{IC} \vee \neg\text{IN}\right] \\ &= \frac{(2^n - q_\pi - 1)!}{2^n!} = \frac{(2^n - q_\pi)!}{2^n} \cdot \frac{1}{2^n - q_\pi} = \Pr[Y_1 = \tau].\end{aligned}$$

Putting Lemmas 8 and 9 together yields the result.

Finally, these two lemmas complete the proof of the security of next. □

C Proof of Preserving Security

Proof. Lemma 2 Formally, we adapt the preserving security game, so that the intermediate state s_d is chosen uniformly at random rather than calculated using the adversarial inputs.

Let $\mathcal{A}$ be the adversary playing in the preserving security game, $\tau'_{\mathcal{A}}$ be as above; the set of adversarial queries but restricted to those made in the first part of the game, before the adversary has submitted her inputs and such that $\|\tau'_{\mathcal{A}}\| = q'_\pi \leq q_\pi$. Let $\mathrm{I}_1, \ldots, \mathrm{I}_d$ be the r-bit adversarial refresh inputs.

Let $\mathsf{Pres}_{\mathsf{Rev}^\pi}$ be the real world preserving security game as defined in Definition 8 with the defined algorithms of $\mathsf{Reverie}$ and chosen permutation π. Let $\mathsf{Pres}'_{\mathsf{Rev}^\pi}$ be identical to $\mathsf{Pres}_{\mathsf{Rev}^\pi}$ except s_d is replaced with $s_d \xleftarrow{\$} \{0,1\}^n$.

We now aim to prove, in two parts that in the real world case, the first two games act the same with a small bound, while in the ideal world case they are identical. Following this, what remains is to prove that the advantage of an adversary in distinguishing the ideal world from the real world in Pres' is precisely the security bound of the next function from Lemma 1. For clarity, we say that $\mathsf{Pres}_{\mathsf{Rev}^\pi}(\mathcal{A}) \implies 1$ means the adversary outputs 1 as her guess of b.

First,

Lemma 10. *For **Game 0** and **Game 1** as described above,*

$$\left|\Pr\left[\mathsf{Pres}^{\mathcal{A}}_{\mathsf{Rev}^\pi} \implies 1 \mid b = 0\right] - \Pr\left[\mathsf{Pres}'^{\mathcal{A}}_{\mathsf{Rev}^\pi} \implies 1 \mid b = 0\right]\right| \leq \frac{q'_\pi + d}{2^n} + \frac{(d+1)(2q'_\pi + d)}{2^n}.$$

Proof. To begin, we note that $s_0 \xleftarrow{\$} \{0,1\}^n$, and is not revealed to the adversary. With this in mind, using lazy sampling of the permutation π, we have

$$\Pr\left[\exists i \in [q'_\pi] s.t. x_i = s_0 \oplus ((\mathrm{I}_1 \oplus \mathsf{seed}_1)\|0^c)\right] \leq \frac{q'_\pi}{2^{n-1}}.$$

Provided that this does not happen, the first intermediary state of the adversarial refreshes will be an unassigned value s_1 which will be uniformly chosen over the remaining $2^n - q'_\pi$ unassigned values of π and thus the probability that the next call to π will be on an already assigned value will be $\frac{q'_\pi+1}{2^{n-1}}$. Iterating this method and we obtain:

$$\frac{q'_\pi}{2^{n-1}} + \frac{q'_\pi + 1}{2^{n-1}} + \cdots + \frac{q'_\pi + d}{2^{n-1}} = \frac{d(2q'_\pi + (d+1))}{2^n}.$$

So with probability $1 - \frac{q'_\pi}{2^{n-1}} + \frac{d(2q'_\pi+(d+1)}{2^n} = 1 - \frac{(d+1)(2q'_\pi+d)}{2^n}$, the resulting state s_d after the adversarial refreshes will be the result of π called on an unassigned

state. Then s_d will be chosen uniformly from the remaining $2^n - q'_\pi - d$ unassigned values.

Finally, this implies the statistical distance between s_d in $\mathsf{Pres}_{\mathsf{Rev}^\pi}$ and s_d in $\mathsf{Pres}'_{\mathsf{Rev}^\pi}$ is at most $\frac{q'_\pi + d}{2^n}$, which together with the previous probability, yields the result.

Next, construct an adversary $\mathcal{A}'$ that runs $\mathcal{A}$ and simulates the Pres' game while inserting its own challenge and outputting the same bit as $\mathcal{A}$, which yields

$$\left|\Pr\left[\mathsf{Pres}^{\mathcal{A}}_{\mathsf{Rev}^\pi} \implies 1 \mid b = 0\right] - \Pr\left[\mathsf{Pres}'^{\mathcal{A}}_{\mathsf{Rev}^\pi} \implies 1 \mid b = 1\right]\right| \leq \mathsf{Adv}^{\mathsf{dist}}_{\mathcal{A}'}(\mathsf{next}^\pi(U_n), (U_n, U_r)).$$

This, together with Lemma 10 completes the proof. □

D Proof of Recovering Security

Proof. Lemma 3 To formalise this, we require the construction of two adversaries, $\mathcal{A}_1, \mathcal{A}_2$ with the former being a $(q_\pi + 1)$-adversary in for the extraction lemma of [13] and the latter, a $\bar{q}_\pi$-adversary in the next distinguishing game. Then we have,

$$\mathsf{Adv}^{(\gamma^*, \mathsf{q}_\pi)-\mathsf{rec}}_{\mathsf{Rev}^\pi_{\mathsf{s,n,r}}}(\mathcal{A}, \mathcal{D}) \leq \mathsf{Adv}^{(\gamma^*, \mathsf{q}_\mathcal{D})-\mathsf{ext}}_{\mathsf{Sp}_{\mathsf{n,r,s}}}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{dist}}_{\mathcal{A}_2}(\mathsf{next}^\pi(U_n), (U_n, U_r)). \quad \text{(D.1)}$$

Let $\mathcal{A}$ be the normal recovering security adversary, then $\mathcal{A}_1$ is built by running $\mathcal{A}$ on $\mathsf{seed}, \gamma_1, \ldots, \gamma_{q_\mathcal{D}}, z_1, \ldots, z_{q_\mathcal{D}}$ received from the challenger, $\mathcal{A}_1$ forwards any $\pi^\pm$ queries from $\mathcal{A}$ to the π oracle, along with any $\mathsf{get} - \mathsf{refresh}$ oracle queries to the associated oracle. Once this has been done, $\mathcal{A}$ will output it's chosen pair (s_0, d), which $\mathcal{A}_1$ will again forward to the challenger as its chosen pair. The challenger will then return the challenge s'_d and the remaining $\mathrm{I}_{k+d+1}, \ldots, \mathrm{I}_{q_\mathcal{D}}$ to $\mathcal{A}_1$, which forwards the latter straight to $\mathcal{A}$ along with the output of $\mathsf{next}(s'_d)$ which it computes. $\mathcal{A}_1$ continues to forward any $\pi^\pm$ queries that $\mathcal{A}$ makes, before $\mathcal{A}$ it's guess b^* which $\mathcal{A}_1$ forwards to the challenger as its own guess. Since $\mathcal{A}_1$ only forwards the queries $\mathcal{A}$ makes to $\pi^\pm$ together with calling $\mathsf{next}(s'_d)$, the query complexity of $\mathcal{A}_1$ is $q_\pi + 1$. It's easy to see that for $b = 0$, this simulates precisely the recovering security game, while $b = 1$ corresponds to $\mathcal{A}$ receiving $(S, T) \leftarrow \mathsf{next}(U_n)$ as opposed to the correct challenge $(S, T) \xleftarrow{\$} (U_n, U_r)$. This is considered in the second term of Equation (D.1). $\mathcal{A}_2$ is now constructed by simulating the $b = 1$ version of the extraction game, while running $\mathcal{A}_1$ and using the distinguishing challenge.

Finally, all that is left is to upper bound these advantages; [13, Lemma 6] yields

$$\mathsf{Adv}^{(\gamma^*, \mathsf{q}_\mathcal{D})-\mathsf{ext}}_{\mathsf{Sp}_{\mathsf{n,r,s}}}(\mathcal{A}_1) \leq \epsilon_{\mathsf{ext}}(q_\pi + 1, q_\mathcal{D}) + \mathsf{Adv}^{(\gamma^*, \mathsf{q}_\mathcal{D})-\mathsf{hit}}_{\mathcal{D},\mathsf{n}}(\mathcal{A}_1),$$

where the latter value is precisely the probability that $\mathcal{A}_1$ queries $\pi^{-1}(s_d)$ in the ideal case. Since $\mathcal{A}_1$ is only making queries to π that $\mathcal{A}$ makes, this is in fact the probability that $\mathcal{A}$ queries $\pi^{-1}(s_d)$ and since $\mathcal{A}$ would either have to guess this value with probability $\frac{q_\pi}{2^{n-1}}$ or have to invert the next challenge to have made this query, this is in fact the advantage of $\mathcal{A}_2$ playing the distinguishing game on the next function, albeit with $\bar{q}_\pi$ queries, due to the queries by the distribution sampler.

Thus, by Lemma 1 we have

$$\epsilon_{\mathsf{next}}(\bar{q}_\pi) \leq \left(5 - \frac{1}{2^r}\right) \frac{\bar{q}_\pi}{2^{c-1}},$$

and

$$\epsilon_{\mathsf{ext}}(q_\pi + 1) \leq \frac{\bar{q}_\pi}{2^{\gamma^*}} + \frac{Q(q_\mathcal{D})}{2^{sr}} + \frac{7(\bar{q}_\pi^2 + 2\bar{q}_\pi + 1)}{2^c} + \frac{(\bar{q}_\pi + 1)q_\mathcal{D} + q_\mathcal{D}^2}{2^{n-1}},$$

which completes the proof. □