The proceedings version of this paper appears in CRYPTO 2016. This is a preliminary full version.

Backdoors in Pseudorandom Number Generators: Possibility and Impossibility Results

Jean Paul Degabriele[1], Kenneth G. Paterson[1], Jacob C. N. Schuldt[2], Joanne Woodage[1]

[1] Royal Holloway, University of London,
[2] AIST, Tokyo

Abstract. Inspired by the Dual EC DBRG incident, Dodis et al. (Eurocrypt 2015) initiated the formal study of backdoored PRGs, showing that backdoored PRGs are equivalent to public key encryption schemes, giving constructions for backdoored PRGs (BPRGs), and showing how BPRGs can be "immunised" by careful post-processing of their outputs. In this paper, we continue the foundational line of work initiated by Dodis et al., providing both positive and negative results.

We first revisit the backdoored PRG setting of Dodis et al., showing that PRGs can be *more strongly* backdoored than was previously envisaged. Specifically, we give efficient constructions of BPRGs for which, given a single generator output, Big Brother can recover the initial state and, therefore, *all* outputs of the BPRG. Moreover, our constructions are *forward-secure* in the traditional sense for a PRG, resolving an open question of Dodis et al. in the negative.

We then turn to the question of the effectiveness of backdoors in robust PRNGs with input (c.f. Dodis et al., ACM-CCS 2013): generators in which the state can be regularly refreshed using an entropy source, and in which, provided sufficient entropy has been made available since the last refresh, the outputs will appear pseudorandom. The presence of a refresh procedure might suggest that Big Brother could be defeated, since he would not be able to predict the values of the PRNG state backwards or forwards through the high-entropy refreshes. Unfortunately, we show that this intuition is not correct: we are also able to construct robust PRNGs with input that are backdoored in a backwards sense. Namely, given a single output, Big Brother is able to rewind through a number of refresh operations to earlier "phases", and recover all the generator's outputs in those earlier phases.

Finally, and ending on a positive note, we give an impossibility result: we provide a bound on the number of previous phases that Big Brother can compromise as a function of the state-size of the generator: smaller states provide more limited backdooring opportunities for Big Brother.

1 Introduction

Background: In the wake of the Snowden revelations, the cryptographic research community has begun to realise that it faces a more powerful and insidious adversary than it had previously envisaged: Big Brother, an adversary willing to subvert cryptographic standards and implementations in order to gain an advantage against users of cryptography. The Dual EC DRBG debacle, and subsequent research showing the widespread use of this NIST-standardised pseudorandom generator (PRG) and its security consequences [11], has highlighted that inserting backdoors into randomness-generating components of systems is a profitable, if high-risk, strategy for Big Brother.

The threat posed by the Big Brother adversary brings new research challenges, both foundational and applied. The study of subversion of cryptographic systems — how to undetectably and securely subvert them, and how to defend against subversion — is a central one. Current research efforts to understand various forms of subversion include the study of Algorithm Substitution Attacks (ASAs) [6,13,24,29,2] and that of backdooring of cryptosystems [11,14,8,3]. These lines of research have a long and rich history through topics such as kleptography [35] and subliminal channels [32]. In an ASA, the subversion is specific to a specific *implementation* of a particular algorithm or scheme, whereas in backdooring, the backdoor resides in the specification of the scheme or primitive itself and any implementation faithful to the specification will be equally vulnerable. There is a balancing act at play with these two types of attack: while ASAs are arguably easier to carry out, their impact is limited to a specific implementation, whereas the successful introduction of a backdoor into a cryptographic scheme, albeit ostensibly harder to mount and subsequently conceal, can have much wider impact.

The importance of randomness: Many cryptographic processes rely heavily on good sources of randomness, for example, key generation, selection of IVs for encryption schemes and random challenges in authentication protocols, and the selection of Diffie-Hellman exponents. Indeed randomness failures of various kinds have led to serious vulnerabilities in widely deployed cryptographic systems, with a growing literature on such failures [19,10,1,26,34,28,23,21,7]. Furthermore it is well established in the theory of cryptography that the security of most cryptographic tasks relies crucially on the quality of that randomness [15].

Since true random bits are hard to generate without specialised hardware, and such hardware has only recently started to become available on commodity computing platforms,[3] Pseudorandom Generators (PRGs) and Pseudorandom Number Generators with input ("PRNGs with input" for short) are almost universally used in implementations. These generate pseudorandom bits instead of truly random bits; PRNGs with input can also have their state regularly refreshed with fresh entropy, though from a possibly biased source of randomness. Typically, a host operating system will make PRNGs with input available to applications, with the entropy being gathered from a variety of events, e.g. keyboard or disk timings, or timing of interrupts and other system events; programming libraries typically also provide access to PRG functionality, though of widely varying quality.

[3] See for example `https://en.wikipedia.org/wiki/RdRand` for a description of Intel's "Bull Mountain" random number generator.

Backdooring Randomness: Given the ubiquity of PRGs and PRNGs with input in cryptographic implementations, they constitute the ideal target for maximising the spread and impact of backdoors. This was probably the rationale behind the Dual EC DRBG [11] which is widely believed to have been backdoored by the NSA. Despite this generator's low-speed, known output biases, and known capability to be backdoored (which was pointed out as early as 2007 by Shumow and Ferguson [31]), it managed to be covertly deployed in a range of widely used systems. Such systems continue to be discovered today, more than three years after the original Snowden revelations relating to Dual EC DRBG and project Bullrun.[4] The Dual EC DRBG provides a particularly useful backdoor to Big Brother: given a single output from the generator, its state can be recovered, and all future outputs can be recovered (with moderate computational effort). Protocols like SSL/TLS directly expose PRG outputs in protocol messages, making the Dual EC DRBG exploitable in practice [11].

Formal analysis of backdoored PRGs: The formal study of backdoored PRGs (BPRGs) was initiated by Dodis et. al. [14], building on earlier work of Vazirani and Vazirani [33]. Dodis et al. showed that BPRGs are equivalent to public-key encryption (PKE) with pseudorandom ciphertexts (IND$-CPA-security), provided constructions using PKE schemes and KEMs, and analysed folklore immunisation techniques. Understanding the nature of backdoored primitives together with their capabilities and limitations is an important first step towards finding solutions that will safeguard against backdooring attacks. For instance the equivalence of BPRGs with public key encryption shown in [14] suggests that a PRG based on purely symmetric techniques is less likely to contain a backdoor, since we currently do not know how to build public key encryption from one-way functions.

A basic question that was posed – and partly answered – in [14] is: *to what extent can a PRG be backdoored while at the same time being provably secure?* This question makes perfect sense in the context of subversion via backdooring, where the backdoor resides in the specification of the PRG itself, and where the PRG can be publicly assessed and its security evaluated. The Dual EC DRBG has notable biases which directly rule out any possibility of it being provably secure as a PRG. Nevertheless, in [14] it is noted that by using special encodings of curve points as in [25,36,9], these biases can be eliminated and the Dual EC DRBG can be turned into a provably forward-secure PRG under the DDH assumption.

Yet the backdoor in the Dual EC DRBG, while relatively powerful and certainly completely undermining security in certain applications like SSL/TLS, has its limitations. In particular, it does not allow Big Brother (who holds the

[4] See for example `http://www.realworldcrypto.com/rwc2016/program/rwc16-shacham.pdf?attredirects=0&d=1` for the Dual EC DRBG being used as a backdoor in Juniper networking equipment; see also `http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security` for the original reporting on project Bullrun.

backdoor key) to predict *previous* outputs from a given output but only future ones. The random-seek BPRG construction of [14] provides a stronger type of backdoor: given any single output, it allows Big Brother to recover any past or future output with probability roughly $\frac{1}{4}$. But the random-seek BPRG construction of [14] attains this stronger backdooring at the expense of no longer being a forward-secure PRG (in the usual sense). Indeed, forward-security and the random-seek backdoor property would intuitively seem to be opposing goals, and it is then natural to ask whether this tradeoff is inherent, or whether strong forms of backdooring of forward-secure PRGs *are* possible. If the limitation was inherent, then a proof of forward-security for a PRG would serve to preclude backdoors with the backward-seek feature, so a forward-secure PRG would be automatically immunised, to some extent, against backdoors.

1.1 Our Contributions

In this work we advance understanding of backdoored generators in two distinct directions.

Stronger backdooring of PRGs: We settle the above open question from [14] in the negative by providing two different constructions of random-seek BPRGs that are provably forward-secure. In fact we demonstrate something substantially stronger:

- Firstly, both of our constructions allow Big Brother to succeed with probability 1 (rather than the 1/4 attained for the random-seek BPRG construction of [14]).
- Secondly, the backdooring is much stronger, in that for both of our BPRG constructions, Big Brother is able to recover the initial state of the BPRG, given only a single output value. This then enables all states and output values to be reconstructed.

Our constructions require a number of cryptographic tools. Unsurprisingly, given the connection between BPRGs and PKE with pseudorandom ciphertexts that was shown in [14], they both make use of the latter primitive. To give a flavour of what lies ahead, we remark that our simplest construction, shown in Figure 7, uses such a PKE scheme to encrypt its state s, with the resulting ciphertext C forming the generator's output; s is also evolved using a one-way function, to provide forward security. Clearly, Big Brother, with access to a single output and the decryption key, can recover the state s. But we use a *trapdoor* one-way function so that Big Brother can then "unwind" s back to its starting value. For the security proof, we need to use a random oracle applied to s to generate the encryption randomness, making our construction reminiscent of the "Encrypt-with-hash" construction of [5], while for technical reasons, we require the trapdoor one-way function to be lossy [27]. Our second construction is in the standard model and combines, in novel ways, other primitives such as re-randomizable PKE schemes.

Backdooring PRNGs with input: We then turn our attention to the study of backdoored PRNGs with input (BPRNGs). This is a very natural extension to the study of BPRGs conducted in [14] and continued here, particularly in view of the widespread deployment of PRNGs with input in real systems.

The formal study of PRNGs with input (but without backdooring) commenced with Barak and Halevi's work in [4], later extended in [16,18]. Various security notions have been proposed in the literature for PRNGs with input, namely *resilience*, *forward security*, *backward security* and *robustness*. Of these, robustness is the strongest notion. It captures the ability of a generator to both preserve security when its entropy inputs are influenced by an attacker and to recover security after its state is compromised, via refreshing (provided sufficient entropy becomes available to it). Robustness is generally accepted as the *de facto* security target for any new PRNG design, though several widely-deployed PRNGs fail to meet it (see, for example, [16,12]).

Given that we are in the backdooring setting for subversion, in which the full specification of the cryptographic primitive targeted for backdooring is public, any construction can be vetted for security. It is therefore logical to require any BPRNG to be robust. (This is analogous to requiring a BPRG to be forward-secure, or at least, a PRG in the traditional sense.) As such, a BPRNG *cannot* just ignore its entropy inputs and revert to being a PRG. One might then hope that, with additional high entropy inputs being used to refresh the generator state, and with this entropy not being under the direct control of Big Brother (since, otherwise, no security at all is possible), backdooring a PRNG with input might be impossible. This would be a positive result in the quest to defeat backdooring. Unfortunately, we show that this is not the case.

As a warm-up, we show how to adapt the robust PRNG of [16] to make it backdoored. This requires only a simple trick (and some minor changes to the processing of entropy): replace the PRG component of the generator with a BPRG. Given a single output from the generator, this then allows Big Brother to compute *all* outputs from the last refresh operation to the next refresh operation. Yet the generator is still robust.

Much more challenging is to develop a robust PRNG with input in which Big Brother can use his backdoor to "pass through" refresh operations when computing generator outputs. We provide a construction which does just that, see Figure 11. Our construction is based on the idea of interleaving outputs of a (non-backdoored) PRNG with encryptions of snapshots of that PRNG's state, using an IND$-CPA secure encryption scheme to ensure pseudorandomness of the outputs. By taking a snapshot of the state whenever it is refreshed and storing a list of the previous k snapshots in the state (for a parameter k), the construction enables Big Brother to recover, with some probability, old output values that were computed as many as k refreshes previously. The actual construction is considerably more complex than this sketch hints, since achieving robustness, in the sense of [16], is challenging when the state has this additional structure. We also sketch variants of this construction that trade state and output size for strength of backdooring.

An impossibility result for BPRNGs: We close the paper on a more positive note, providing an impossibility result showing that backdooring in a strong sense cannot be achieved (whilst preserving robustness) without significantly enlarging the state of the generator. More precisely, we show that it is not possible for Big Brother to perform a *state* recovery attack in which he recovers more than some number k of properly refreshed previous states from an output of the generator, when k is large relative to the state-size of the BPRNG. A precise formalisation of our result is contained in Theorem 5.

Note that the backdooring attack here requires more of Big Brother than might be needed in practice, since he may be considered successful if he can recover just one previous state, or a fraction of the previous BPRNG outputs. Our construction shows that backdooring of this kind is certainly possible. Nor does our result say anything about Big Brother's capabilities (or lack thereof) when it comes to recovering *future* states/outputs (after a generator has undergone further high-entropy refresh operations). It is an important open problem to strengthen our impossibility results – and to improve our constructions – to explore the limits of backdooring for PRNGs with input.

2 Preliminaries

2.1 Notation

The set of binary strings of length n is denoted $\{0,1\}^n$ and ε denotes the empty string. For any two binary strings x and y we write $|x|$ to denote the size of x and $x\|y$ to denote their concatenation. For any set U we denote by $u \twoheadleftarrow U$ the process of sampling an element uniformly at random from U and assigning it to u. All logs are to base 2.

2.2 Entropy

We recall a number of standard definitions on entropy, statistical distance, and (k,ϵ)-extractors in Appendix A.1.

Definition 1. *An (k,ϵ)-extractor* $\mathsf{Ext} : \{0,1\}^* \times \{0,1\}^v \to \{0,1\}^w$ *is said to be online-computable on inputs of length p if there exists a pair of efficient algorithms* $\mathsf{iterate} : \{0,1\}^p \times \{0,1\}^p \times \{0,1\}^v \to \{0,1\}^p$*, and* $\mathsf{finalize} : \{0,1\}^p \times \{0,1\}^v \to \{0,1\}^w$ *such that for all inputs $\bar{I} = (I_1, \ldots, I_d)$ where each $I_j \in \{0,1\}^p$, and $d \geq 2$, then after setting $y_1 = I_1$, and $y_j = \mathsf{iterate}(y_{j-1}, I_j; A)$ $j = 2, \ldots, d$, it holds that*

$$\mathsf{Ext}(\bar{I}; A) = \mathsf{finalize}(y_d; A).$$

2.3 Cryptographic Primitives

In Appendix A.2, we recall a number of standard definitions for PKE schemes. Throughout this work we require that PKE schemes be length-regular.

For the constructions that follow, we shall require an IND\$-CPA-secure PKE scheme; that is to say a PKE scheme having pseudorandom ciphertexts. We define such schemes formally below. Concrete and efficient examples of such schemes can be obtained by applying carefully constructed encoding schemes to the group elements of ciphertexts in the ElGamal encryption scheme (in which ciphertexts are of the form $(g^R, M \cdot g^{Rx})$ where g generates a group of prime order p in which DDH is hard; $(g^x, x) \leftarrow \mathsf{KGen}$ with $x \twoheadleftarrow \mathbb{Z}_p$; $R \twoheadleftarrow \mathbb{Z}_p$; and M is a message, encoded here as a group element); see for example [25,36,9].

Definition 2. *A PKE scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be (t, q, δ)-IND\$-CPA-secure if for all adversaries $\mathscr{A}$ running in time t and making at most q oracle queries, it holds that $\mathsf{Adv}_{\mathcal{E}}^{\text{ind\$-cpa}}(\mathscr{A}) \leq \delta$, where:*

$$\mathsf{Adv}_{\mathcal{E}}^{\text{ind\$-cpa}}(\mathscr{A}) = \Big| \Pr\Big[(pk, sk) \leftarrow \mathsf{KGen} : \mathscr{A}^{\mathsf{Enc}(pk,\cdot)}(pk) \Rightarrow 1\Big] \\ - \Pr\Big[(pk, sk) \leftarrow \mathsf{KGen} : \mathscr{A}^{\$(\cdot)}(pk) \Rightarrow 1\Big] \Big|$$

and $\$(\cdot)$ is such that on input a message M it returns a random string of size $|\mathsf{Enc}(pk, M)|$.

It is straightforward to show that if $\mathcal{E}$ is (t, q, δ)-IND\$-CPA-secure, then it is also $(t, q, 2\delta)$-IND-CPA-secure in the usual sense.

We shall also utilise PKEs which are *statistically re-randomizable*; again the ElGamal scheme and its group-element-encoded variants have the required property.

Definition 3. *[20] A (t, q, δ, ν)-statistically re-randomizable encryption scheme is a tuple of algorithms $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Rand}, \mathsf{Dec})$ where $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is a standard PKE scheme and Rand is an efficient randomised algorithm such that for all $(pk, sk) \leftarrow \mathsf{KGen}$ and for all M, R_0',*

$$\Delta(\{\mathsf{Enc}(pk, M; R_0) : R_0 \twoheadleftarrow \mathsf{Coins}(\mathsf{Enc})\}, \\ \{\mathsf{Rand}(\mathsf{Enc}(pk, M; R_0'); R_1) : R_1 \twoheadleftarrow \mathsf{Coins}(\mathsf{Rand}) :\}) \leq \nu.$$

That is, the distributions of an honestly generated ciphertext and a ciphertext obtained by applying Rand to one generated with arbitrary randomness are statistically close. We write $\mathsf{Rand}(C_0; R_1, \ldots, R_q)$ to denote the value of C_q where $C_j = \mathsf{Rand}(C_{j-1}; R_j)$ for $j = 1, \ldots, q$.

We now define encryption schemes which have the additional property of being *reverse re-randomizable*. It is easy to see that ElGamal encryption and its encoded variants has the required property.

Definition 4. *A (t, q, δ, ν)-statistically reverse re-randomizable encryption scheme $\mathcal{E}$ is a tuple of algorithms $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Rand}, \mathsf{Rand}^{-1}, \mathsf{Dec})$ such that:*

- *$(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Rand}, \mathsf{Dec})$ is a (t, q, δ, ν) statistically re-randomizable encryption scheme.*

- Rand^{-1} *is an efficient algorithm such that for all* $(pk, sk) \leftarrow \mathsf{KGen}$ *and for all* M,R_0,R_1, *it holds that, if* $C = \mathsf{Enc}(pk, M; R_0)$, *then:*

$$\Pr\left[\mathsf{Rand}^{-1}(\mathsf{Rand}(C; R_1); R_1) = C\right] = 1.$$

Suppose $C_q = \mathsf{Rand}(C_0; R_1, \ldots, R_q)$, *so that* $C_j = \mathsf{Rand}(C_{j-1}; R_j)$ *for* $j = 1, \ldots, q$. *Then, from the above, we know that* $C_{j-1} = \mathsf{Rand}^{-1}(C_j; R_j)$ *for* $1 \leq j \leq q$*; to denote* C_0*, we write* $\mathsf{Rand}^{-1}(C_q; R_1, \ldots, R_q)$.

We recall the definitions of trapdoor one-way permutations, and lossy trapdoor permutations, in Appendix A.2.

2.4 Pseudorandom Generators

A pseudorandom generator (PRG) takes a small amount of true statistical randomness as an input seed, and outputs arbitrary (polynomial) length bit-strings which are *pseudorandom*. Following [14], we will equip PRGs with a parameter generation algorithm, setup. This allows backdooring to be introduced into the formalism.

Definition 5. *A* PRG *is a triple of algorithms* $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$*, with associated parameters* $(n, l) \in \mathbb{N}^2$*, defined as follows:*

- $\mathsf{setup} : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ *takes random coins as input and outputs a pair of parameters* (pp, bk)*, where* pp *denotes the public parameter for the generator, and* bk *is the secret backdoor parameter. In a non-backdoored* PRG*, we set* $bk = \perp$.
- $\mathsf{init} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^n$ *takes* pp *and random coins as input, and returns an initial state for the* PRG*,* $s_0 \in \{0,1\}^n$.
- $\mathsf{next} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^l \times \{0,1\}^n$ *takes* pp *and a state* $s \in \{0,1\}^n$ *as input, and outputs an output/state pair* $(r, s') \leftarrow \mathsf{next}(pp, s)$ *where* $r \in \{0,1\}^l$ *is the* PRG*'s output, and* $s' \in \{0,1\}^n$ *is the updated state.*

Definition 6. *Let* $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *be a* PRG*. Given an initial state* s_0*, we set* $(r_i, s_i) \leftarrow \mathsf{next}(pp, s_{i-1})$ *for* $i = 1, \ldots, q$*. We write* $\mathsf{out}^q(\mathsf{next}(pp, s_0))$ *for the sequence of outputs* $r_1, \ldots, r_q$ *and* $\mathsf{state}^q(\mathsf{next}(pp, s_0))$ *for the sequence of states* $s_1, \ldots, s_q$ *produced by this process.*

Definition 7 (PRG **Security).** *Let* $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *be a* PRG*. Consider the game* $\text{PRG-DIST}^{\mathscr{A},q}_{\mathsf{PRG}}$ *of Figure 1 in which the adversary receives either* q *outputs from the* PRG *or* q *random strings of the appropriate size. We define the* PRG *distinguishing advantage of* $\mathscr{A}$ *against* PRG *to be*

$$\mathsf{Adv}^{\text{dist}}_{\mathsf{PRG}}(\mathscr{A}, q) = 2|\Pr\left[\text{PRG-DIST}^{\mathscr{A},q}_{\mathsf{PRG}} \Rightarrow \mathsf{true}\right] - \frac{1}{2}|.$$

Definition 8. *A* PRG $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *is said to be* (t, q, δ)*-secure if for all adversaries* $\mathscr{A}$ *running in time at most* t *it holds that* $\mathsf{Adv}^{\text{dist}}_{\mathsf{PRG}}(\mathscr{A}, q) \leq \delta$.

```
Game $\text{PRG-DIST}^{\mathscr{A},q}_{\mathsf{PRG}}$
$(pp, bk) \twoheadleftarrow \mathsf{setup}$
$s_0 \twoheadleftarrow \mathsf{init}(pp)$
$r^0_1, \ldots, r^0_q \leftarrow \mathsf{out}^q(\mathsf{next}(pp, s_0))$
$r^1_1, \ldots, r^1_q \twoheadleftarrow (\{0,1\}^l)^q$
$b \twoheadleftarrow \{0,1\}$
$b' \leftarrow \mathscr{A}(pp, r^b_1, \ldots, r^b_q)$
return $(b = b')$
```

```
Game $\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}}$
$(pp, bk) \twoheadleftarrow \mathsf{setup}$
$s_0 \twoheadleftarrow \mathsf{init}(pp)$
$r^0_1, \ldots, r^0_q \leftarrow \mathsf{out}^q(\mathsf{next}(pp, s_0))$
$r^1_1, \ldots, r^1_q \twoheadleftarrow (\{0,1\}^l)^q$
$s_1, \ldots, s_q \leftarrow \mathsf{state}^q(\mathsf{next}(pp, s_0))$
$b \twoheadleftarrow \{0,1\}$
$b' \leftarrow \mathscr{A}(pp, r^b_1, \ldots, r^b_q, s_q)$
return $(b = b')$
```

Fig. 1: The games for $\text{PRG-DIST}^{\mathscr{A},q}_{\mathsf{PRG}}$ and $\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}}$.

Definition 9 (PRG Forward Security). *Let* $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *be a* PRG*. Consider the game* $\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}}$ *of Figure 1 in which the adversary receives either q outputs from the* PRG *and the final state, or q random strings of the appropriate size and the final state. We define the* PRG *forward-security advantage of $\mathscr{A}$ against* PRG *to be*

$$\mathsf{Adv}^{\mathrm{fwd}}_{\mathsf{PRG}}(\mathscr{A}, q) := 2|\Pr\left[\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}} \Rightarrow \mathsf{true}\right] - \frac{1}{2}|.$$

Definition 10. *A* PRG PRG *is said to be* (t, q, δ)-FWD*-secure if for all adversaries $\mathscr{A}$ running in time at most t it holds that* $\mathsf{Adv}^{\mathrm{fwd}}_{\mathsf{PRG}}(\mathscr{A}, q) \leq \delta$*.*

2.5 Backdoored Pseudorandom Generators

The first formal treatment of backdoored PRGs was that of Dodis et al. [14]. Intuitively, a backdoored cryptosystem is a scheme coupled with some secret backdoor information. In the view of an adversary who does not know the backdoor information, the scheme fulfils its usual security definition. However an adversary in possession of the backdoor information will gain some advantage in breaking the security of the cryptosystem. The backdoor attacker is modelled as an algorithm which we call $\mathscr{B}$ (for 'Big Brother'), to distinguish it from an attacker $\mathscr{A}$ whose goal is to break the usual security of the scheme *without* access to the backdoor. Whilst the backdoor attacker $\mathscr{B}$ will be external in the sense that it will only be able to observe public outputs and parameters, the attack is also internalised as the backdoor algorithm is designed alongside, and incorporated into, the scheme.

We define backdoored PRGs (BPRGs) in conjunction with different games $\text{BPRNG-TYPE}^{\mathscr{B},q}_{\mathsf{PRG}}$ which capture specific backdooring goals, each game having a corresponding advantage term. The three games considered in [14] are defined in Figure 2.

```
Game BPRG-DIST$^{\mathscr{B},q}_{\mathsf{PRG}}$
    $(pp, bk) \twoheadleftarrow \mathsf{setup}$
    $s_0 \twoheadleftarrow \mathsf{init}(pp)$
    $r_1^0, \ldots, r_q^0 \leftarrow \mathsf{out}^q(\mathsf{next}(pp, s_0))$
    $r_1^1, \ldots, r_q^1 \twoheadleftarrow (\{0,1\}^l)^q$
    $b \twoheadleftarrow \{0,1\}$
    $b^* \leftarrow \mathscr{B}(pp, bk, r_1^0, \ldots, r_q^0)$
    return $(b = b^*)$
```

```
Game BPRG-NEXT$^{\mathscr{B},q}_{\mathsf{PRG}}$
    $(pp, bk) \twoheadleftarrow \mathsf{setup}$
    $s_0 \twoheadleftarrow \mathsf{init}(pp)$
    $r_1^0, \ldots, r_q^0 \leftarrow \mathsf{out}^q(\mathsf{next}(pp, s_0))$
    $s_1, \ldots, s_q \leftarrow \mathsf{state}^q(\mathsf{next}(pp, s_0))$
    $s_q^* \leftarrow \mathscr{B}(pp, bk, r_1^0, \ldots, r_q^0)$
    return $(s_q = s_q^*)$
```

```
Game BPRG-RSEEK$^{\mathscr{B},q}_{\mathsf{PRG}}(i, j)$
    $(pp, bk) \twoheadleftarrow \mathsf{setup}$
    $s_0 \twoheadleftarrow \mathsf{init}(pp)$
    $r_1^0, \ldots, r_q^0 \leftarrow \mathsf{out}^q(\mathsf{next}(pp, s_0))$
    $s_1, \ldots, s_q \leftarrow \mathsf{state}^q(\mathsf{next}(pp, s_0))$
    $r_j^* \leftarrow \mathscr{B}(pp, bk, i, j, r_i)$
    return $(r_j = r_j^*)$
```

Fig. 2: Security games for backdooring of PRGs.

Definition 11. *A tuple of algorithms* $\overline{\mathsf{PRG}} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathscr{B})$ *is defined to be a* $(t, q, \delta, (\mathit{type}, \epsilon))$*-secure* BPRG *if:*

- $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *is a* (t, q, δ)*-secure PRG;*
- $\mathsf{Adv}^{\mathrm{type}}_{\mathsf{PRG}}(\mathscr{B}, q) \geq \epsilon$.

Definition 12. *Let* $\overline{\mathsf{PRG}} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathscr{B})$ *be a* BPRG*. We define*

- $\mathsf{Adv}^{\mathrm{dist}}_{\mathsf{PRG}}(\mathscr{B}, q) := 2|\Pr\left[\text{BPRG-DIST}^{\mathscr{B},q}_{\mathsf{PRG}} \Rightarrow \mathsf{true}\right] - \frac{1}{2}|$,
- $\mathsf{Adv}^{\mathrm{next}}_{\mathsf{PRG}}(\mathscr{B}, q) := \Pr\left[\text{BPRG-NEXT}^{\mathscr{B},q}_{\mathsf{PRG}} \Rightarrow \mathsf{true}\right]$,
- $\mathsf{Adv}^{\mathrm{rseek}}_{\mathsf{PRG}}(\mathscr{B}, q) := \min_{1 \leq i,j, \leq q} \Pr\left[\text{BPRG-RSEEK}^{\mathscr{B},q}_{\mathsf{PRG}}(i, j) \Rightarrow \mathsf{true}\right]$.

In Figure 2, game BPRG-DIST$^{\mathscr{B},q}_{\mathsf{PRG}}$ challenges Big Brother to use the backdoor to break the security of the PRG in the most basic sense of distinguishing real from random outputs. In game BPRG-NEXT$^{\mathscr{B},q}_{\mathsf{PRG}}$, $\mathscr{B}$ aims to recover the current state of the PRG given q consecutive outputs from the generator. This is a far more powerful compromise since it then allows $\mathscr{B}$ to predict all of the generator's future outputs. In the third game, BPRG-RSEEK$^{\mathscr{B},q}_{\mathsf{PRG}}(i, j)$, $\mathscr{B}$ is given only the i^{th} output (rather than q outputs) and index j, and tries to recover the j^{th} output (but not any state).

It is noted in [14] that an adversary $\mathscr{B}$ winning in game BPRG-NEXT$_{\mathsf{PRG}}^{\mathscr{B},q}$ represents a stronger form of backdooring than an adversary $\mathscr{B}$ winning in game BPRG-DIST$_{\mathsf{PRG}}^{\mathscr{B},q}$ for the same parameters, whilst an adversary $\mathscr{B}$ winning in game BPRG-RSEEK$_{\mathsf{PRG}}^{\mathscr{B},q}(i,j)$ may be more or less powerful than one for game BPRG-NEXT$_{\mathsf{PRG}}^{\mathscr{B},q}$ depending on the circumstances. The paper [14] presents constructions of BPRGs that are backdoored in the BPRG-NEXT$_{\mathsf{PRG}}^{\mathscr{B},q}$ and BPRG-RSEEK$_{\mathsf{PRG}}^{\mathscr{B},q}(i,j)$ senses, but does also note that their construction for a scheme of the latter type is *not* forward-secure.

Both for their intrinsic interest, and because they will be needed in our later constructions of backdoored PRNGs with input, we are interested in BPRGs that *are* forward secure against normal adversaries. For a generic type of game BPRNG-TYPE$_{\mathsf{PRG}}^{\mathscr{B},q}$, these are formally defined as follows.

Definition 13. *A tuple of algorithms* $\overline{\mathsf{PRG}} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathscr{B})$ *is said to be a* $(t, q, \delta, (\mathit{type}, \epsilon))$-FWD-*secure* BPRG *if:*

- $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *is a* (t, q, δ)-FWD-*secure PRG;*
- $\mathsf{Adv}_{\mathsf{PRG}}^{\mathrm{type}}(\mathscr{B}, q) \geq \epsilon$.

2.6 Pseudorandom Number Generators with Input

Definition 14 (PRNG **with input).** *A* PRNG *with input is a tuple of algorithms* $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{init}, \mathsf{refresh}, \mathsf{next})$ *with associated parameters* $(n, l, p) \in \mathbb{N}^3$*, where:*

- $\mathsf{setup} : \{0,1\}^* \to \{0,1\}^*$ *takes as input some random coins and returns a public parameter pp.*
- $\mathsf{init} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^n$ *takes the public parameter pp and some random coins to return an initial state* s_0.
- $\mathsf{refresh} : \{0,1\}^* \times \{0,1\}^n \times \{0,1\}^p \to \{0,1\}^n$ *takes as input the public parameter pp, the current state* S*, and a sample* I *from the entropy source, and returns a new state* s'.
- $\mathsf{next} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^l$ *takes as input the public parameter pp and the current state* s*, and returns a new state* s' *together with an output string* r.

Definition 15 (Distribution Sampler). *A distribution sampler* $\mathscr{D} : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^p \times \mathbb{R}^{\geq 0} \times \{0,1\}^*$ *is a probabilistic and possibly stateful algorithm which takes its current state* σ *as input and returns an updated state* σ'*, a sample* I*, an entropy estimate* γ*, and some leakage information* z *about* I*. The state* σ *is initialised to the empty string.*

A distribution sampler $\mathscr{D}$ *is said to be valid up to* q_r *samples, if for all* $j \in \{1, \ldots, q_r\}$ *it holds (with probability 1) that:*

$$\mathrm{H}_\infty\left(I_j \,|\, I_1, \ldots, I_{j-1}, I_{j+1}, \ldots, I_{q_r}, z_1, \ldots, z_{q_r}, \gamma_1, \ldots, \gamma_{q_r}\right) \geq \gamma_j$$

where $(\sigma_i, I_i, \gamma_i, z_i) = \mathscr{D}(\sigma_{i-1})$ *for* $i \in \{1, \ldots, q_r\}$ *and* $\sigma_0 = \varepsilon$.

```
Game $\mathrm{ROB}^{\mathscr{D},\mathscr{A}}_{\mathsf{PRNG},\gamma^*}$
    $pp \twoheadleftarrow \mathsf{setup}$
    $\sigma \leftarrow \varepsilon;\ c \leftarrow \infty$
    $s \twoheadleftarrow \mathsf{init}(pp)$
    $b \twoheadleftarrow \{0,1\}$
    $b' \leftarrow \mathscr{A}^{\textsc{Ref},\textsc{Ror},\textsc{Get},\textsc{Set}}(pp)$
    return $b' = b$

Ref
    $(\sigma, I, \gamma, z) \twoheadleftarrow \mathscr{D}(\sigma)$
    $s \leftarrow \mathsf{refresh}(pp, s, I)$
    $c \leftarrow c + \gamma$
    return $(\gamma, z)$

Ror
    $(s, r_0) \leftarrow \mathsf{next}(pp, s)$
    $r_1 \twoheadleftarrow \{0,1\}^l$
    if $c < \gamma^*$
        $c \leftarrow 0$
        return $r_0$
    else return $r_b$

Get
    $c \leftarrow 0$
    return $s$

Set $(s^*)$
    $c \leftarrow 0$
    $s \leftarrow s^*$
```

Fig. 3: PRNG with input security game $\mathrm{ROB}^{\mathscr{D},\mathscr{A}}_{\mathsf{PRNG},\gamma^*}$.

2.7 Security for Pseudorandom Number Generators with Input

We now turn to discussing security definitions for PRNGs with input. We follow [16], with some minor differences noted below.

Definition 16 (Security of PRNG with Input). *With references to the security game shown in Figure 3, a* PRNG *with input* $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{init}, \mathsf{refresh}, \mathsf{next})$ *is said to be* $(t, q_r, q_n, q_c, \gamma^*, \epsilon)$-ROB*-secure if, for any distribution sampler* $\mathscr{D}$ *valid up to* q_r *samples, and any adversary* $\mathscr{A}$ *running in time at most* t*, making at most* q_r *queries to* Ref*,* q_n *queries to* Ror *and a total of* q_c *queries to* Get *and* Set*, the corresponding advantage in game* $\mathrm{ROB}^{\mathscr{D},\mathscr{A}}_{\mathsf{PRNG},\gamma^*}$ *is bounded by* ϵ*, where*

$$\mathsf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG}}(\mathscr{A}, \mathscr{D}) := 2|\Pr\left[\mathrm{ROB}^{\mathscr{D},\mathscr{A}}_{\mathsf{PRNG},\gamma^*} \Rightarrow \mathsf{true}\right] - \frac{1}{2}|.$$

Our definition here deviates from that in [16] in the following ways.

- We generalise the syntax so as to allow the state to be initialised according to some arbitrary distribution rather than requiring it to be uniformly random. In particular we allow this distribution to depend on pp. This facilitates our backdooring definitions to follow.
- We have removed the Next oracle from the model, without any loss of generality (as was shown in [12]).

One of the key insights of [16] is to decompose the somewhat complex notion of robustness into the two simpler notions of PRE and REC security. We recall these definitions below, generalised here to include the init algorithm.

Definition 17 (Preserving and Recovering Security). *Consider the security games described in Figure 4. The* PRE *security advantage of an adversary* $\mathscr{A}$ *against a* PRNG *with input* PRNG *is defined to be*

$$\mathsf{Adv}^{pre}_{\mathsf{PRNG}}(\mathscr{A}) := 2|\Pr\left[\mathrm{PRE}^{\mathscr{A}}_{\mathsf{PRNG}} \Rightarrow \mathsf{true}\right] - \frac{1}{2}|.$$

```
Game $\mathrm{PRE}^{\mathscr{A}}_{\mathsf{PRNG}}$
  $b \twoheadleftarrow \{0,1\}$
  $pp \twoheadleftarrow \mathsf{setup}$
  $s^0 \twoheadleftarrow \mathsf{init}(pp)$
  $\mathbf{I}[1:d] \leftarrow \mathscr{A}(pp)$
  for $i = 1$ to $d$
    $s^i \leftarrow \mathsf{refresh}(pp, s^{i-1}, \mathbf{I}[i])$
  $(s_0, r_0) \leftarrow \mathsf{next}(pp, s^d)$
  $s_1 \twoheadleftarrow \mathsf{init}(pp);\ r_1 \twoheadleftarrow \{0,1\}^l$
  $b' \leftarrow \mathscr{A}(pp, s_b, r_b)$
  return $b' = b$
```

```
Game $\mathrm{REC}^{\mathscr{D},\mathscr{A},q_r}_{\mathsf{PRNG},\gamma^*}$
  $k \leftarrow 0;\ \sigma[0] \leftarrow \varepsilon$
  $b \twoheadleftarrow \{0,1\}$
  $pp \twoheadleftarrow \mathsf{setup}$
  for $i = 1$ to $q_r$
    $(\boldsymbol{\sigma}[i], \mathbf{I}[i], \boldsymbol{\gamma}[i], \mathbf{z}[i]) \twoheadleftarrow \mathscr{D}(\boldsymbol{\sigma}[i-1])$
  $(s^0, d) \leftarrow \mathscr{A}^{\textsc{Sam}}(pp, \boldsymbol{\gamma}, \mathbf{z})$
  for $i = k+1$ to $k+d$
    $s^i \leftarrow \mathsf{refresh}(pp, s^{i-1}, \mathbf{I}[k+i])$
  $(s_0, r_0) \leftarrow \mathsf{next}(pp, s^d)$
  $s_1 \twoheadleftarrow \mathsf{init}(pp);\ r_1 \twoheadleftarrow \{0,1\}^l$
  $b' \leftarrow \mathscr{A}(pp, \mathbf{I}[k+d+1:q_r], s_b, r_b)$
  return $b' = b$
```

```
$\textsc{Sam}$
  $k \leftarrow k+1$
  return $\mathbf{I}[k]$
```

Fig. 4: PRNG with input security games $\mathrm{PRE}^{\mathscr{A}}_{\mathsf{PRNG}}$ and $\mathrm{REC}^{\mathscr{D},\mathscr{A},q_r}_{\mathsf{PRNG},\gamma^*}$.

The REC *security advantage with respect to parameters q_r, γ^* of an adversary/sampler pair ($\mathscr{A}$, $\mathscr{D}$) against a* PRNG *with input* PRNG *is defined to be*

$$\mathsf{Adv}^{rec}_{\mathsf{PRNG}}(\mathscr{A}, \mathscr{D}) := 2|\Pr\left[\mathrm{REC}^{\mathscr{D},\mathscr{A},q_r}_{\mathsf{PRNG},\gamma^*} \Rightarrow \mathsf{true}\right] - \frac{1}{2}|.$$

In the REC *security game, it is required that the $\boldsymbol{\gamma}[i]$ values output by $\mathscr{D}$ and the value d output by $\mathscr{A}$ satisfies $\sum_{j=k+1}^{k+d} \boldsymbol{\gamma}[j] \geq \gamma^*$.*

Definition 18 (Preserving Security). *A* PRNG *with input* PRNG *is said to have (t, ϵ_{pre})-*PRE *security if for all attackers $\mathscr{A}$ running in time t, it holds that $\mathsf{Adv}^{pre}_{\mathsf{PRNG}}(\mathscr{A}) \leq \epsilon_{pre}$.*

Definition 19 (Recovering Security). *A* PRNG *with input* PRNG *is said to have $(t, q_r, \gamma^*, \epsilon_{rec})$-*REC *security if for any attacker $\mathscr{A}$ and sampler $\mathscr{D}$ valid up to q_r samples and running in time t, it holds that $\mathsf{Adv}^{rec}_{\mathsf{PRNG}}(\mathscr{A}, \mathscr{D}) \leq \epsilon_{rec}$.*

Informally, preserving security concerns a generator's ability to maintain security (in the sense of having pseudorandom state and output) when the adversary completely controls the entropy source used to refresh the generator but does not compromise its state. Meanwhile, recovering security captures the idea that a generator whose state is set by the adversary should eventually get to a secure state, and start producing pseudorandom outputs, once sufficient entropy has been made available to it. The proof of Theorem 1 is given in Appendix A.3.

Theorem 1. *Let* PRNG *be a* PRNG *with input. If* PRNG *has both (t, ϵ_{pre})-*PRE *security, and $(t, q_r, \gamma^*, \epsilon_{rec})$-*REC *security, then* PRNG *is $((t', q_r, q_n, q_c), \gamma^*, \epsilon)$-*ROB *secure where $t \approx t'$ and $\epsilon = q_n(\epsilon_{pre} + \epsilon_{rec})$.*

```
$\mathsf{evolve}(\mathsf{PRNG}, pp, s, \boldsymbol{rp}, \mathscr{D})$

**parse** $\boldsymbol{rp}$ **as** $(a_1, b_1, \ldots, a_\rho, b_\rho)$
$\mathcal{S} \leftarrow (); \sigma \leftarrow \varepsilon$
**for** $i = 1$ **to** $\rho$
  **for** $j = 1$ **to** $a_i$
    $(r, s) \leftarrow \mathsf{next}(pp, s)$
    $\mathcal{S} \leftarrow \mathcal{S} \| (r, s)$
  **for** $k = 1$ **to** $b_i$
    $(\sigma, I, \gamma, z) \leftarrow \mathscr{D}(\sigma)$
    $s \leftarrow \mathsf{refresh}(pp, s, I)$
**return** $\mathcal{S}$
```

Fig. 5: The evolve algorithm.

To simplify notation, we will make use of an algorithm, evolve, to generate output values and update the internal state of a PRNG. It takes as input a PRNG with input $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathsf{refresh})$, public parameter pp, an initial state s, a refresh pattern $\boldsymbol{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$, and a distribution sampler $\mathscr{D}$. The refresh pattern $\boldsymbol{rp}$ denotes a sequence of calls to next and refresh; for each i, a_i denotes the number of consecutive calls to next and b_i denotes the subsequent number of consecutive calls to refresh. More specifically, evolve proceeds as shown in Figure 5.

The output of evolve is a sequence, $(r_1, s_1, \ldots, r_{q_n}, s_{q_n})$, of PRNG output and state pairs, where $q_n = \sum_{i=1}^{\rho} a_i$. Based on evolve, we define an additional algorithm, out, which takes the same input, runs evolve, and returns only the output values $(r_1, \ldots, r_{q_n})$.

3 Stronger Models and New Constructions for Backdoored Pseudorandom Generators

In this section, we first present two new, strong backdooring security models for PRGs. The stronger of the two implies all the backdooring notions in [14]. We then give two new constructions of BPRGs which achieve our two backdooring notions. In contrast to the strongest constructions in [14], all of our constructions are *forward-secure*.

3.1 Backdoored PRG Security Models

In the first of our two new models, the BPRG is run with initial state s_0 to produce q outputs $r_1, \ldots, r_q$. The Big Brother adversary $\mathscr{B}$ is then given a particular output r_i, and challenged to recover the initial state s_0 of the BPRG.

```
Game BPRG-FIRST$^{\mathscr{B},q}_{\overline{\mathsf{PRG}}}(i)$
$(pp, bk) \twoheadleftarrow \mathsf{setup}$
$s_0 \twoheadleftarrow \mathsf{init}(pp)$
$r_1, \dots, r_q \leftarrow \mathsf{out}^q(\mathsf{next}(pp, s_0))$
$s_0^* \leftarrow \mathscr{B}(pp, bk, r_i)$
return $(s_0 = s_0^*)$
```

```
Game BPRG-OUT$^{\mathscr{B},q}_{\overline{\mathsf{PRG}}}(i)$
$(pp, bk) \twoheadleftarrow \mathsf{setup}$
$s_0 \twoheadleftarrow \mathsf{init}(pp)$
$r_1, \dots, r_q \leftarrow \mathsf{out}^q(\mathsf{next}(pp, s_0))$
$r_1^*, \dots, r_q^* \leftarrow \mathscr{B}(pp, bk, r_i)$
return $((r_1, \dots r_q) = (r_1^*, \dots, r_q^*))$
```

Fig. 6: Backdoored PRG security games BPRG-FIRST and BPRG-OUT.

In the second model, the BPRG is again run with initial state s_0 to produce q outputs, one of which is given to $\mathscr{B}$. However $\mathscr{B}$ is now asked to reproduce the remaining $q-1$ unseen output values. We formalise these two models as games BPRG-FIRST and BPRG-OUT in Figure 6.

Definition 20. *Let* $\overline{\mathsf{PRG}} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathscr{B})$ *be a* BPRG. *We define*

- $\mathsf{Adv}^{\text{first}}_{\overline{\mathsf{PRG}}}(\mathscr{B}, q, i) := \Pr\left[\text{BPRG-FIRST}^{\mathscr{B},q}_{\overline{\mathsf{PRG}}}(i) \Rightarrow \mathsf{true}\right]$, *and*
- $\mathsf{Adv}^{\text{out}}_{\overline{\mathsf{PRG}}}(\mathscr{B}, q, i) := \Pr\left[\text{BPRG-OUT}^{\mathscr{B},q}_{\overline{\mathsf{PRG}}}(i) \Rightarrow \mathsf{true}\right]$

Discussion We observe that our first backdooring notion, as formalised in BPRG-FIRST$^{\mathscr{B},q}_{\overline{\mathsf{PRG}}}$ and $\mathsf{Adv}^{\text{first}}_{\overline{\mathsf{PRG}}}(\mathscr{B}, q, i)$, is strictly stronger than the three notions for BPRGs defined in [14] and discussed in Section 2.5: it is straightforward to see that any $(t, q, \delta, (\mathsf{first}, \epsilon))$-secure BPRG is also a $(t, q, \delta, (\mathsf{type}, \epsilon))$-secure BPRG for $\mathsf{type} \in \{\mathsf{dist}, \mathsf{state}, \mathsf{rseek}\}$.

Moreover, simple comparison of definitions shows that any $(t, q, \delta, (\mathsf{out}, \epsilon))$-secure BPRG is also a $(t, q, \delta, (\mathsf{type}, \epsilon))$-secure BPRG for $\mathsf{type} \in \{\mathsf{dist}, \mathsf{rseek}\}$. However, a BPRG backdoored in the out sense need not be backdoored in the state sense, since the latter concerns state prediction rather than output prediction. (And indeed it is easy to construct separating examples for the out and state backdooring notions.)

Since the initial state of a PRG determines all of its output, it is also clear that any $(t, q, \delta, (\mathsf{first}, \epsilon))$-secure BRPG is also a $(t, q, \delta, (\mathsf{out}, \epsilon))$-secure BPRG. However, the converse need not hold, and first backdooring is strictly stronger than out backdooring. To see this, consider $\overline{\mathsf{PRG}}$, a $(t, q, \delta, (\mathsf{out}, \epsilon))$-secure BPRG, and define a modified BRPG $\overline{\mathsf{PRG}}'$ in which the initial state s_0 is augmented to $s_0||d$ for $d \twoheadleftarrow \{0,1\}^n$, but where d is not used in any computations and all other algorithms of $\overline{\mathsf{PRG}}$ are left unchanged. In particular, the output produced by $\overline{\mathsf{PRG}}'$ is identical to that of $\overline{\mathsf{PRG}}$. Then it is easy to see that $\overline{\mathsf{PRG}}'$ is a $(t, q, \delta, (\mathsf{out}, \epsilon))$-secure BPRG, but that $\mathsf{Adv}^{\text{first}}_{\overline{\mathsf{PRG}}}(\mathscr{B}, q, i) \leq 2^{-n}$, since $\mathscr{B}$ can do no better than guessing the n extra bits of state d.

In most attack scenarios, and taking Big Brother's perspective, the ability of $\mathscr{B}$ to compute all unseen output (as in out) is as useful in practice as being able to compute the initial state (as in first), since it is the output values of the BPRG that will be consumed in applications. This makes the out notion a natural and powerful target for constructions of BPRGs. That said, in the sequel we will obtain constructions for the even stronger first setting.

A $(t, q, \delta, (\mathsf{rseek}, \epsilon))$-secure BPRG is also a $(t, q, \delta, (\mathsf{out}, \epsilon^{q-1}))$-secure BPRG, implying an exponential loss in going from rseek backdooring to out backdooring. This means that achieving either first or out backdooring with a high value of ϵ is significantly more powerful than achieving rseek backdooring with the same ϵ.

3.2 Forward-secure BPRGs in the Random Oracle Model

We present our first construction for a forward-secure BPRG that is backdoored in the first sense in Figure 7. This construction uses as ingredients an LTDP family and an IND\$-CPA-secure PKE scheme. Its security analysis is in the Random Oracle Model (ROM). It achieves our strongest first notion with $\epsilon = 1$.

The scheme is reminiscent of the "Encrypt-with-Hash" paradigm for constructing deterministic encryption schemes from [5]. At each stage, the generator encrypts its own state s, with randomness derived from hashing s, to produce the next output. The IND\$-CPA-security of the PKE scheme ensures these outputs are pseudorandom. The state s is also transformed by applying a one-way function F at each stage. This is necessary to provide forward security against non-$\mathscr{B}$ adversaries. The function is trapdoored, enabling $\mathscr{B}$ to decrypt an output to recover a state, then reverse the state update repeatedly to recover the initial state, thereby realising first backdooring. For technical reasons that will become apparent in the proof, we require the one-way function F to be a lossy permutation. The proof of the following theorem is in Appendix B.1.

Theorem 2. *Let* $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a* (t, q, δ)*-IND\$-CPA secure PKE scheme. Let* $\mathsf{LTDP} = (\mathsf{G}_0, \mathsf{G}_1, \mathsf{S}, \mathsf{F}, \mathsf{F}^{-1})$ *be a family of* (n, k, t, ϵ)*-lossy trapdoor permutations. Then* $\overline{\mathsf{PRG}} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathscr{B})$ *with algorithms as shown in Figure 7 is a* $(t', q, (2\delta + 3\epsilon + (q+1)2^{-(k-1)}), (\mathsf{first}, 1))$*-FWD secure* BPRG *in the ROM, where* $t' \approx t$.

3.3 Standard Model, Forward-secure BPRGs from Reverse Re-randomizable Encryption

Our second construction dispenses with the ROM and the use of lossy trapdoor permutations, at the expense of requiring as a component an IND\$-CPA-secure reverse re-randomizable PKE scheme (see Definition 4). It is instantiable in the standard model using a variant of the ElGamal encryption scheme. The scheme is again backdoored in the first sense with $\epsilon = 1$.

The scheme, shown in Figure 8, uses algorithm next' from a normal (forward-secure) PRG PRG' to generate the next state s' and a pseudorandom value t

```
setup
    $(pk, sk) \leftarrow \mathsf{KGen}$
    $(\mathsf{PK}, \mathsf{SK}) \twoheadleftarrow \mathsf{G}_1$
    $pp \leftarrow (pk, \mathsf{PK})$
    $bk \leftarrow (sk, \mathsf{SK})$
    return $(pp, bk)$
```

```
init $(pp)$
    $(pk, \mathsf{PK}) \leftarrow pp$
    $s_0 \twoheadleftarrow \mathsf{S}(\mathsf{PK})$
    return $(s_0)$
```

```
next $(pp, s)$
    $(pk, \mathsf{PK}) \leftarrow pp$
    $r \leftarrow \mathsf{Enc}(pk, s; \mathsf{RO}(s))$
    $s' \leftarrow \mathsf{F}_{\mathsf{PK}}(s)$
    return $(r, s')$
```

```
$\mathscr{B}(bk, i, r_i)$
    $(sk, \mathsf{SK}) \leftarrow bk$
    $s^*_{i-1} \leftarrow \mathsf{Dec}(sk, r_i)$
    $s^*_0 \leftarrow \mathsf{F}_{\mathsf{SK}}^{-(i-1)}(s^*_{i-1})$
    return $(s^*_0)$
```

Fig. 7: Construction of a forward-secure BPRG $(\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathscr{B})$ from an LTDP family $\mathsf{LTDP} = (\mathsf{G}_0, \mathsf{G}_1, \mathsf{S}, \mathsf{F}, \mathsf{F}^{-1})$ and an IND\$-CPA-secure PKE scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$.

using the current state s as a seed. The value t is then used to re-randomise a ciphertext C that encrypts an initial state value s_0, and the 'old' value C is used as the generator's output r. The re-randomisation at each step ensures that the outputs collectively appear pseudorandom to a regular PRG adversary; the fact that PRG' is forward-secure ensures that the constructed BPRG is too.

Meanwhile, the use of PKE allows $\mathscr{B}$ (who knows the decryption key) to recover s_0 from any of the generator's outputs, run the component generator PRG' from its starting state s_0, and recover all the values t used for re-randomisation at each step; finally $\mathscr{B}$ can run the re-randomisation process backwards to recover the initial state. The proof of the following theorem is in Appendix B.2.

Theorem 3. *Let* $\mathcal{E} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Rand}, \mathsf{Rand}^{-1}, \mathsf{Dec})$ *be a* (t, q, δ, ν)-IND\$-CPA *secure reverse re-randomizable encryption scheme, and suppose that* $\mathsf{PRG}' = (\mathsf{setup}', \mathsf{init}', \mathsf{next}')$ *is a* (t, q, ϵ_{fwd})*-secure* PRG*. Then* $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathscr{B})$ *as defined in Figure 8 is a* $(t', q, 6\delta + 2\epsilon_{fwd} + q(q+3)\nu/2, (\mathsf{first}, 1))$-FWD *secure* BPRG*, where* $t' \approx t$.

4 Backdooring PRNGs with Input

In this section, we address the second main theme in our paper: backdooring of PRNGs with input. To begin with, we show a simple construction for a PRNG with input that is both robust and subject to a limited form of backdooring: given a single output, $\mathscr{B}$ can recover the state and all outputs back to the previous $\mathsf{refresh}$ and up to the next $\mathsf{refresh}$ operations (see Section 4.1). We then move on to provide our formal definition for backdoored PRNGs with input (BPRNGs) in Section 4.2; this definition demands much more of $\mathscr{B}$, asking him to compute outputs beyond $\mathsf{refresh}$ operations, at the same time as asking that the BPRNG remain robust. Finally, in Section 4.3, we give a construction for a BPRNG meeting our backdooring notion for PRNGs with input, with various extensions to this construction being described in Section 4.4.

```
setup
  $(pk, sk) \leftarrow \mathsf{KGen}$
  $(pp', \bot) \twoheadleftarrow \mathsf{setup}'$
  $pp \leftarrow (pk, pp')$
  $bk \leftarrow sk$
  return $(pp, bk)$

init $(pp)$
  $(pk, pp') \leftarrow pp$
  $s_0 \twoheadleftarrow \mathsf{init}'(pp')$
  $C_0 \twoheadleftarrow \mathsf{Enc}_{pk}(s_0)$
  $S \leftarrow (s_0, C_0)$
  return $S$
```

```
next $(pp,\ S)$
  $(pk, pp') \leftarrow pp$
  $(s, C) \leftarrow S$
  $(t, s') \leftarrow \mathsf{next}'(pp', s)$
  $C' \leftarrow \mathsf{Rand}(C, t)$
  $r \leftarrow C$
  $S \leftarrow (s', C')$
  return $(r, S)$
```

```
$\mathscr{B}(sk, r_i, i)$
  $C^*_{i-1} \leftarrow r_i$
  $s^*_0 \leftarrow \mathsf{Dec}_{sk}(C_{i-1})$
  $(t^*_1, \ldots, t^*_q) \leftarrow \mathsf{out}^q(\mathsf{next}'(pp', s^*_0))$
  for $j = 1, \ldots, i-1$
    $C^*_{j-1} \leftarrow \mathsf{Rand}^{-1}(C^*_j, t^*_j)$
  $S^* \leftarrow (s^*_0, C^*_0)$
  return $(S^*)$
```

Fig. 8: Construction of a forward-secure BPRG $(\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathscr{B})$ from a (t, q, δ, ν)-reverse-re-randomizable IND\$-CPA-secure PKE scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ and a forward-secure PRG $\mathsf{PRG}' = (\mathsf{setup}', \mathsf{init}', \mathsf{next}')$.

4.1 A Simple Backdoored PRNG

Let $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{init}, \mathsf{refresh}, \mathsf{next})$ be a ROB-secure PRNG with input. By considering the special case of Game $\mathrm{ROB}^{\mathscr{D},\mathscr{A}}_{\mathsf{PRNG},\gamma^*}$ in which the adversary $\mathscr{A}$ makes no Set or Ref calls, and one Get call at the conclusion of the game, it is straightforward to see that $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ must be a FWD-secure PRG. This suggests that in order to backdoor PRNG, we might try to replace PRG with a BPRG. As long as this implicit BPRG is running without any refreshes, this should enable $\mathscr{B}$ to carry out backdooring.

To make this idea concrete, we present in Figure 9 a construction of a ROB-secure PRNG with input from a PRG PRG. This scheme is closely based on the PRNG with input from [16]. It utilises an online-computable extractor and a FWD-secure PRG; our main modification is to ensure that repeated next calls are processed via a repeated iteration of a FWD-secure PRG. A proof of robustness for this PRNG with input is easily derived from that of the original construction:

Lemma 1. *Let* $\mathsf{Ext} : \{0,1\}^* \times \{0,1\}^v \rightarrow \{0,1\}^n$ *be an online-computable* $(\gamma^*, \epsilon_{ext})$*-extractor. Let* $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *be a* (t, q, ϵ_{prg})*-PRG such that* $s_0 \twoheadleftarrow \mathsf{init}(pp)$ *is equivalent to* $s_0 \twoheadleftarrow \{0,1\}^n$*. Then* $\mathsf{PRNG} = (\overline{\mathsf{setup}}, \overline{\mathsf{init}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}})$ *as shown in Figure 9 is a* $((t', q_r, q_n, q_c), \gamma^*, q_n(2\epsilon_{prg} + q_r^2\epsilon_{ext} + 2^{-n+1}))$*-robust PRNG with input, where* $t' \approx t$*.*

We now simply substitute a FWD-secure BPRG (such as that presented in Theorem 2) for PRG in this construction. Now, during the period between

```
$\overline{\mathsf{setup}}$
  $(pp', \perp) \twoheadleftarrow \mathsf{setup}$
  $\mathrm{A} \twoheadleftarrow \{0,1\}^v$
  $pp \leftarrow (pp', \mathrm{A})$
  $bk \leftarrow \perp$
  return $(pp, bk)$

$\overline{\mathsf{init}}(pp)$
  $s_1 \twoheadleftarrow \{0,1\}^n$
  $s_2 \leftarrow 0^p$
  $\mathsf{flgRfrsh} \leftarrow 0$
  $\overline{s} \leftarrow (s_1, s_2, \mathsf{flgRfrsh})$
  return $(\overline{s})$
```

```
$\overline{\mathsf{refresh}}(pp, s, I)$
  parse $pp$ as $(pp', \mathrm{A})$
  parse $\overline{s}$ as $(s_1, s_2, \mathsf{flgRfrsh})$
    $s_2 \leftarrow \mathsf{iterate}(s_2, I; \mathrm{A})$
    $\mathsf{flgRfrsh} \leftarrow 1$
  $\overline{s} \leftarrow (s_1, s_2, \mathsf{flgRfrsh})$
  return $(\overline{s})$
```

```
$\overline{\mathsf{next}}(pp, s)$
  parse $pp$ as $(pp', \mathrm{A})$
  parse $\overline{s}$ as $(s_1, s_2, \mathsf{flgRfrsh})$
  if $\mathsf{flgRfrsh} = 1$
    $U \leftarrow \mathsf{finalize}(s_2; \mathrm{A})$
    $s_1 \leftarrow U \oplus s_1$
    $s_2 \leftarrow 0^p$
  $(s_1, r) \leftarrow \mathsf{next}(pp'; s_1)$
  $\mathsf{flgRfrsh} \leftarrow 0$
  $\overline{s} \leftarrow (s_1, s_2, \mathsf{flgRfrsh})$
  return $(\overline{s}, r)$
```

Fig. 9: Construction of a robust PRNG $\overline{\mathsf{PRNG}}$ from a FWD-secure PRG PRG, based on [16].

any pair of refresh calls in which the PRNG is producing output, we inherit the backdooring advantage of the BPRG in the new construction. However, the effectiveness of this backdoor is highly limited: as soon as refresh is called, the state of the PRNG is refreshed with inputs, which, if of sufficiently high entropy, will make the state information-theoretically unpredictable. Then $\mathscr{B}$ would need to compromise more output in order to regain his backdooring advantage.

One implication of this construction is that it makes it clear that, when considering stronger forms of backdooring, we must turn our attention to subverting refresh calls in some way.

4.2 Formal Definition for Backdoored PRNGs with Input

To make our backdooring models for PRNGs with input as strong as possible, we wish to make minimal assumptions about Big Brother's influence, whilst allowing the non-backdoored adversary $\mathscr{A}$, to whom the backdoored schemes must still appear secure, maximum power to compromise the scheme. To this end, we will model $\mathscr{B}$ as a passive observer who is able to capture just one PRNG output, which he is then challenged to exploit. Simultaneously, we demand that the scheme is still secure in the face of a ROB-adversary $\mathscr{A}$, with all the capabilities this allows. Notably, the latter condition also offers the benefit of allowing us to explore the extent to which a guarantee of robustness may act as an immuniser against backdooring.

In our models to follow, we do not allow $\mathscr{B}$ any degree of compromise over the distribution sampler $\mathscr{D}$. This is again to fit with our ethos of making minimal assumptions on $\mathscr{B}$'s capabilities. It strengthens the backdooring model by

```
Game BPRNG-STATE$^{\mathscr{B}}_{\overline{\mathsf{PRNG}},\mathscr{D}}(\boldsymbol{rp}, i, j)$
$(pp, bk) \twoheadleftarrow \mathsf{setup}$
$s_0 \twoheadleftarrow \mathsf{init}(pp)$
$(r_1, s_1, \ldots, r_{q_n}, s_{q_n})$
        $\leftarrow \mathsf{evolve}(\overline{\mathsf{PRNG}}, pp, s_0, \boldsymbol{rp}, \mathscr{D})$
$s'_j \leftarrow \mathscr{B}(pp, bk, r_i, i, j, \boldsymbol{rp})$
return $(s'_j = s_j)$
```

```
Game BPRNG-OUT$^{\mathscr{B}}_{\overline{\mathsf{PRNG}},\mathscr{D}}(\boldsymbol{rp}, i, j)$
$(pp, bk) \twoheadleftarrow \mathsf{setup}$
$s_0 \twoheadleftarrow \mathsf{init}(pp)$
$(r_0, \ldots, r_{q_n}) \leftarrow \mathsf{out}(\overline{\mathsf{PRNG}}, pp, s_0, \boldsymbol{rp}, \mathscr{D})$
$r'_j \leftarrow \mathscr{B}(pp, bk, r_i, i, j, \boldsymbol{rp})$
return $(r'_j = r_j)$
```

Fig. 10: Backdooring security games BPRNG-STATE$^{\mathscr{B}}_{\overline{\mathsf{PRNG}},\mathscr{D}}$ and BPRNG-OUT$^{\mathscr{B}}_{\overline{\mathsf{PRNG}},\mathscr{D}}$ for BPRNGs.

demanding that the backdoor be effective against *all* samplers $\mathscr{D}$ valid up to q_r samples, including in particular those not under the control of $\mathscr{B}$. We also note that, in the extreme case where $\mathscr{B}$ has complete knowledge of all the inputs used in refresh calls, then $\mathscr{B}$'s view of the evolution of the state is deterministic and the PRNG is reduced to a FWD-secure PRG which is periodically reseeded with correlated values. Thus this restriction on Big Brother's power ensures a clear separation between the results of Section 3 and those which follow.

Next consider a PRNG with input which produces its output via a sequence of refresh and next calls. The evolution of the state, and subsequent production of output, is determined not only by the number of such calls, but also by their position in the sequence. To reflect this, each backdooring game below will take as input the specific refresh pattern $\boldsymbol{rp}$ which was used to produce the challenge. In line with this, and to reflect the fact that the refresh pattern may impact $\mathscr{B}$'s ability to subvert the scheme, the advantage of $\mathscr{B}$ in our formal definition will be allowed to depend on the refresh pattern $\boldsymbol{rp}$.

We present two new backdooring models for PRNGs with input in Figure 10. In the first game, the PRNG is evolved according to the specified refresh pattern. Big Brother is given an output r_i, and challenged to recover state s_j. In the second game, Big Brother is again given output r_i, but now we ask him to recover a different output value r_j. In both games, Big Brother is additionally given the refresh pattern. Stronger notions can be achieved by considering games in which Big Brother is not given the refresh pattern, but for simplicity, we will consider the games shown in Figure 10. In Section 4.4 we will discuss how our concrete construction of a BPRNG presented in Section 4.3 can be extended to the stronger setting in which Big Brother is not given the used refresh pattern. As with the corresponding PRG definitions in Section 3.1, a BPRNG backdoored in the state sense is strictly stronger than one backdoored in the out sense.

Definition 21. *A tuple of algorithms* $\overline{\mathsf{PRNG}} = ($setup, init, next, refresh, $\mathscr{B})$ *is said to be a* $(t, q_r, q_n, q_c, \gamma^*, \epsilon, (\mathit{type}, \delta))$*-robust BPRNG, where* $\mathit{type} \in \{\mathit{state}, \mathit{out}\}$*, if*

- $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{init}, \mathsf{refresh}, \mathsf{next})$ *is a* $(t, q_r, q_n, q_c, \gamma^*, \epsilon)$*-robust* PRNG *with input;*
- *For all refresh patterns* $\boldsymbol{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$*, where* a_i, b_i, n *are polynomial in the security parameter, for all distribution samplers* $\mathscr{D}$*, for all* $1 \leq i, j \leq \sum_{\nu=1}^{\rho} a_\nu$*, where* $i \neq j$*, it holds that* $\mathsf{Adv}^{\text{type}}_{\mathsf{PRNG},\mathscr{D}}(\boldsymbol{rp}, i, j) \geq \delta(\boldsymbol{rp}, i, j)$ *where*

$$\mathsf{Adv}^{\text{type}}_{\mathsf{PRNG},\mathscr{D}}(\boldsymbol{rp}, i, j) := \Pr\left[\text{BPRNG-TYPE}^{\mathscr{B}}_{\mathsf{PRNG},\mathscr{D}}(\boldsymbol{rp}, i, j) \Rightarrow \mathsf{true}\right].$$

We note that by replacing the index j with a vector of indices $(j_1, \ldots, j_k)$, we can immediately extend both of the above games to challenge Big Brother to recover multiple outputs and states.

4.3 Backdoored PRNG Construction

In Figure 11, we present our construction of a BPRNG. The construction makes use of an ordinary non-backdoored PRNG with input, PRNG, and is based on the simple idea of interleaving outputs of PRNG with encryptions of snapshots of the state of PRNG, using an IND\$-CPA secure encryption scheme. By taking a snapshot of the state whenever this is refreshed and storing a list of the previous k snapshots, the construction will enable $\mathscr{B}$ to recover, with reasonable probability, the previous output values that were computed up to k refreshes ago. Of course, this means that the state of the final construction is large compared to that of the PRNG with input used as a component in its construction.

More specifically, the construction maintains a list of ciphertexts, $(C_1, \ldots, C_k)$, encrypting k snapshots of the state of PRNG. A snapshot of the state is taken in the $\overline{\mathsf{next}}$ algorithm of our construction, whenever the previous operation was a refresh (see line 6-9 of $\overline{\mathsf{next}}$). This ensures that if the state is successively refreshed multiple times, only a single snapshot will be stored. To produce an output value, the construction will use the next function of PRNG to compute a seed r which will either be used to directly compute an output value $\overline{r}$ via a pair of PRGs, or used to re-randomize $(C_1, \ldots, C_k)$, which will then be used as $\overline{r}$. The combination of the IND\$-CPA-security of the encryption scheme and the re-randomization will ensure that the output value in the latter case will remain pseudorandom to a regular PRNG adversary. Which of the two different output values the construction will produce is decided based on the seed r (see line 11-17 of $\overline{\mathsf{next}}$).

We prove robustness of the generator by going via preserving and recovering security. To be able to achieve these notions, the ciphertexts $(C_1, \ldots, C_k)$ are re-randomized a second time in $\overline{\mathsf{next}}$ to ensure that the overall state returned by $\overline{\mathsf{next}}$ appears independent of the output value $\overline{r}$. Furthermore, to ensure recovering security, in which the adversary is allowed to maliciously set the state, the construction requires that the validity of ciphertexts can be verified. In particular, we assume the used encryption scheme is equipped with an additional algorithm, $\mathsf{invalid}$, which given a public key pk and a ciphertext C, returns 1 if C is *in*valid for pk, and 0 if it is valid. This is used to ensure that the state of the construction always contains valid ciphertexts (see line 3-5 of $\overline{\mathsf{next}}$). Additionally, we require the used encryption scheme to satisfy a stronger re-randomization property than was introduced in Section 2: the re-randomisation of an adversarially chosen ciphertext should be indistinguishable from the encryption of any message. We will formalize this property below.

For the Big Brother algorithm $\mathscr{B}$ in the construction to be successful, it is required that the output value $\overline{r}_i$ given to $\mathscr{B}$ corresponds to $(C_1, \ldots, C_k)$, and that the output value $\overline{r}_j$ that $\mathscr{B}$ is required to recover corresponds to a value computed directly from the then current state of PRNG. Since the type of the produced output is decided from the output of PRNG and a PRG which are both assumed to be good generators, this will happen with probability close to 1/4. Furthermore, it is required that the number of refresh periods between $\overline{r}_j$ and $\overline{r}_i$ is less than k. More precisely, for a refresh pattern $\boldsymbol{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$, the number of refresh periods PRNG has undergone when $\overline{r}_i$ and $\overline{r}_j$ are produced, are $i_{ref} = \max_\sigma[\sum_{\nu=1}^{\sigma} a_\nu < i]$ and $j_{ref} = \max_\sigma[\sum_{\nu=1}^{\sigma} a_\nu < j]$, respectively. If $i_{ref} - j_{ref} < k$, the initial refreshed state used to compute $\overline{r}_j$ will be encrypted in $C_{i_{ref} - j_{ref} + 1}$. Hence, all $\mathscr{B}$ has to do is to decrypt and iterate this state $j_{it} = j - \sum_{\nu=1}^{j_{ref}} a_\nu$ times to obtain the seed used to compute $\overline{r}_j$.

The full construction, shown in Figure 11, is based on a (non-backdoored) (n, l, p)-PRNG with input, $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{init}, \mathsf{refresh}, \mathsf{next})$, a pair of PRGs $\mathsf{PRG} : \{0,1\}^l \rightarrow \{0,1\}^{2ku+1}$ and $\mathsf{PRG}' : \{0,1\}^u \rightarrow \{0,1\}^{k \times m}$, and a re-randomizable encryption scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Rand}, \mathsf{Dec}, \mathsf{invalid})$ with message space $\{0,1\}^n$, randomness space $\{0,1\}^u$, and ciphertext space $\{0,1\}^m$, and produces a $(k \times m + n + 1, k \times m, p)$-PRNG with input.

Before proving the construction to be robust and backdoored, we formalize the stronger re-randomization property mentioned above. Note that this property is not comparable to the re-randomization definition for PKE given in Section 2: that was a statistical notion concerning encryptions of the same message, while, in contrast, the following is a computational notion regarding possibly different messages.

Definition 22. *An encryption scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\{0,1\}^n$ is said to be (t, δ)-*strongly re-randomizable*, if there exists a polynomial time algorithm* Rand *such that*

– *For all $(pk, sk) \leftarrow \mathsf{KGen}$, $M \in \{0,1\}^n$, and $c \leftarrow \mathsf{Enc}(pk, M)$, it holds that*

$$\Pr[\mathsf{Dec}_{sk}(\mathsf{Rand}(C)) = M] = 1$$

.

– *For all adversaries $\mathscr{A}$ with running time t and for all messages $M \in \{0,1\}^n$, it holds that $\mathsf{Adv}_{\mathcal{E}}^{\mathsf{rand}}(\mathscr{A}) < \delta$, where*

$$\mathsf{Adv}_{\mathcal{E}}^{\mathsf{rand}}(\mathscr{A}) = \Big| \Pr\big[(pk, sk) \leftarrow \mathsf{KGen}; b \leftarrow \{0,1\}; C^* \leftarrow \mathscr{A}(pk); \\ C_0 \leftarrow \mathsf{Rand}(pk, C^*); C_1 \leftarrow \mathsf{Enc}(pk, M); b' \leftarrow \mathscr{A}(C_b) : b = b'\big] - 1/2 \Big|.$$

In the above, it is required that the output C^* of $\mathscr{A}$ is a valid ciphertext under pk.

It is relatively straightforward to see that ElGamal encryption satisfies the above re-randomization property. Specifically, for a public key $y = g^x$ and a ciphertext $C = (C^1, C^2) = (g^r, M \cdot y^r)$, a re-randomization C_0 of C is obtained by picking random r' and computing $C_0 = (C^1 \cdot g^{r'}, C^2 \cdot y^{r'})$. However, under the DDH assumption, the tuples $(g, g^{r'}, y, y^{r'})$ and $(g, g^{r'}, y, z)$ are indistinguishable, where z is a random group element. Hence, re-randomization of C is indistinguishable from multiplying the components of C with random group elements, which again makes C_0 indistinguishable from two random group elements. Likewise, the encryption of any message M, $C_1 = (g^r, M \cdot y^r)$, is indistinguishable from two random group elements under the DDH assumption, which makes C_0 and C_1 indistinguishable.

The proof of the following theorem appears in Appendix C.1.

Theorem 4. *Let* PRG *and* PRG' *be* ϵ_{prg}*-secure and* ϵ'_{prg}*-secure PRGs respectively, and let* PRNG *be a* (t, ϵ_{pre})-PRE *and* $(t, q_r, \gamma^*, \epsilon_{rec})$-REC *secure PRNG with input. Suppose further that* $\mathcal{E}$ *is a* $(t, q_{ind}, \epsilon_{ind})$-IND\$-CPA *secure and* (t, ϵ_{rand})*-strongly re-randomizable encryption scheme. Then* $\overline{\mathsf{PRNG}}$ *shown in Figure 11 is a* $(t', q_r, q_n, q_c, \gamma^*, \epsilon, (\mathsf{out}, \delta))$*-robust BPRNG, where* $t' \approx t$*,*

$$\epsilon = 2q_n(8\epsilon_{ind} + 2\epsilon_{prg} + 2\epsilon'_{prg} + 4k\epsilon_{rand} + 3\epsilon_{pre} + \epsilon_{rec})$$

and

$$\delta(\boldsymbol{rp}, i, j) = \begin{cases} (1/4 - 2\epsilon_{prg} - a(\epsilon_{pre} + \epsilon_{rec})) & \textit{if } j \leq i \ \wedge \ i_{ref} - j_{ref} + 1 \leq k \\ 0 & \textit{otherwise} \end{cases}$$

where $\boldsymbol{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$*,* $a = \sum_{\nu=1}^{\rho} a_\nu$*,* $i_{ref} \leftarrow \max_\sigma [\sum_{\nu=1}^{\sigma} a_\nu < i]$*, and* $j_{ref} \leftarrow \max_\sigma [\sum_{\nu=1}^{\sigma} a_\nu < j]$*.*

4.4 Extensions and Modifications of our Main Construction

The above construction can be modified and extended to provide slightly different properties. For example, an alternative to storing a snapshot of a refreshed state by rotating the ciphertexts $(C_1, \ldots, C_k)$ as done in line 9 of $\overline{\mathsf{next}}$, would be to choose a random ciphertext to replace. More specifically, the output value r of PRNG computed in line 7 could be stretched to produce a $\log k$ bit value t, and ciphertext C_t would then be replaced with C_0. Note, however, that $\mathscr{B}$ would no longer be able to tell which ciphertext corresponds to which snapshot of the state. This can be addressed if the used encryption scheme is additionally assumed to be additively homomorphic, e.g. like ElGamal encryption, which, using an appropriate group, also satisfies all of the other requirements of the construction. In this case, the construction would be able to maintain an encrypted counter of the number of refresh periods, and, for each snapshot, store an encrypted value corresponding to the number of refresh periods PRNG has

```
$\overline{\mathsf{setup}}$
1 :  $pp \twoheadleftarrow \mathsf{setup}$
2 :  $(pk, sk) \twoheadleftarrow \mathsf{KGen}$
3 :  $\overline{pp} \leftarrow (pp, pk)$
4 :  $bk \leftarrow sk$
5 :  return $(\overline{pp}, bk)$
```

```
$\overline{\mathsf{init}}(\overline{pp})$
1 :  parse $\overline{pp}$ as $(pp, pk)$
2 :  $s \twoheadleftarrow \mathsf{init}(pp)$
3 :  $C_1 \twoheadleftarrow \mathsf{Enc}(pk, s)$
4 :  for $i = 2$ to $k$
5 :      $C_i \twoheadleftarrow \mathsf{Enc}(pk, 0^n)$
6 :  $\phi \leftarrow 0$
7 :  return $(s, C_1 \dots C_k, \phi)$
```

```
$\overline{\mathsf{refresh}}(\overline{pp}, \overline{S}, I)$
1 :  parse $\overline{pp}$ as $(pp, pk)$
2 :  parse $\overline{S}$ as $(s, C_1 \dots C_k, \phi)$
3 :  $s \leftarrow \mathsf{refresh}(pp, s, I)$
4 :  $\phi \leftarrow 1$
5 :  return $(s, C_1 \dots C_k, \phi)$
```

```
$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
1 :  parse $\overline{pp}$ as $(pp, pk)$
2 :  parse $\overline{S}$ as $(s, C_1 \dots C_k, \phi)$
3 :  for $i = 1$ to $k$
4 :      if $\mathsf{invalid}(pk, C_i)$
5 :          $C_i \leftarrow \mathsf{Enc}(pk, 0^n; 0^u)$
6 :  if $\phi = 1$
7 :      $(s, r) \leftarrow \mathsf{next}(pp, s)$
8 :      $C_0 \leftarrow \mathsf{Enc}(pk, s; r)$
9 :      $C_k \leftarrow C_{k-1}; \dots ; C_1 \leftarrow C_0$
10 : $(s, r) \leftarrow \mathsf{next}(pp, s)$
11 : $(b, r_1, \dots, r_{2k}) \leftarrow \mathsf{PRG}(r)$
12 : if $b = 0$
13 :     $\overline{r} \leftarrow \mathsf{PRG}'(r_1)$
14 : else
15 :     for $i = 1$ to $k$
16 :         $C_i \leftarrow \mathsf{Rand}(C_i, r_i)$
17 :     $\overline{r} \leftarrow (C_1 \dots C_k)$
18 : for $i = 1$ to $k$
19 :     $C_i \leftarrow \mathsf{Rand}(C_i, r_{k+i})$
20 : $\phi \leftarrow 0$
21 : $\overline{S} \leftarrow (s, C_1 \dots C_k, \phi)$
22 : return $(\overline{S}, \overline{r})$
```

```
$\mathscr{B}(\overline{pp}, bk, \overline{r}_i, i, j, \boldsymbol{rp})$
1 :  parse $\overline{pp}$ as $(pp, pk)$
2 :  parse $\boldsymbol{rp}$ as
3 :          $(a_1, b_1, \dots, a_\rho, b_\rho)$
4 :  parse $\overline{r}_i$ as $(C_1 \dots C_k)$
5 :  $i_{ref} \leftarrow \max_{\sigma} \left[\sum_{\nu=1}^{\sigma} a_\nu < i\right]$
6 :  $j_{ref} \leftarrow \max_{\sigma} \left[\sum_{\nu=1}^{\sigma} a_\nu < j\right]$
7 :  if $j > i$ OR $i_{ref} - j_{ref} \geq k$
8 :      return $\perp$
9 :  $s \leftarrow \mathsf{Dec}(bk, C_{(i_{ref} - j_{ref} + 1)})$
10 : $(s_0, r) \leftarrow \mathsf{next}(s)$
11 : $j_{it} \leftarrow j - \sum_{\nu=1}^{j_{ref}} a_\nu$
12 : for $z = 1$ to $j_{it}$
13 :     $(s_z, r_z) \leftarrow \mathsf{next}(pp, s_{z-1})$
14 : $(b, r'_1, \dots, r'_{2k}) \leftarrow \mathsf{PRG}(r_{j_{it}})$
15 : return $\mathsf{PRG}'(r'_1)$
```

Fig. 11: Construction of a robust BPRNG using as components a re-randomisable PKE scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand}, \mathsf{invalid})$, a PRNG with input $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{init}, \mathsf{refresh}, \mathsf{next})$, and PRGs PRG and PRG'.

undergone before the snapshot was taken. If the ciphertexts containing these values are concatenated with $(C_1, \dots, C_k)$ to produce the output value $\overline{r}$, then $\mathscr{B}$ obtains sufficient information to derive what state to use to recover a given output value. This yields a construction with slightly different advantage function $\delta(\boldsymbol{rp}, i, j)$ compared to the above construction; instead of a sharp drop to 0 when i and j are separated by k refresh periods, the advantage gradually declines as the distance (in terms of the number of refresh periods) between i and j increases.

The above construction can furthermore be modified to produce shorter output values. Specifically, instead of setting $\overline{r} = (C_1, \dots, C_k)$ in line 16 of $\overline{\mathsf{next}}$, a random ciphertext C_t could be chosen as $\overline{r}$, by stretching the output of PRG in line 11 with an additional $\log k$ bits to produce t. This will reduce the output length from km bits to m bits. However, a similar problem to the above occurs: $\mathscr{B}$ will not be able to tell which snapshot C_t represents. Using a similar solution to the above will increase the output length to $2m$ bits. This modification will essentially reduce the backdooring advantage by a factor of $1/k$ compared to the above construction.

Lastly, we note that the above construction assumes $\mathscr{B}$ receives as input the refresh pattern $\boldsymbol{rp}$. Again, by maintaining encrypted counters for both the number of refresh periods and the number of produced output values for each snapshot, we can obtain an algorithm $\mathscr{B}$ which does not require $\boldsymbol{rp}$ as input, but at the cost of increasing the output size.

All of the above modifications can be shown to be secure using almost identical arguments to the existing security analysis for the above construction.

5 On the Inherent Resistance of PRNGs with Input to Backdoors

In the previous section we have shown a construction, and variations thereof, for a PRNG with input that is backdoored in a powerful sense: from a given output Big Brother can recover prior state and output values past an arbitrary number of refreshes. One can see however that in our constructions, Big Brother's ability to go past refreshes is limited by the size of the state and output of the constructed generator. We now show that this limitation is inherent in any PRNG with input that is robust.

In particular consider the sequence representing the evolution of a PRNG's state, and select a subsequence of states where any two states are separated by consecutive refreshes that in combination have high entropy. Then we will show that the number of such states that Big Brother can predict *simultaneously* with non-negligible probability is limited by the size of the state. Thus if we limit the state size of a robust PRNG, then Big Brother's ability in exploiting any potential backdoors that it may contain must *decrease* as more entropy becomes available to the PRNG.

5.1 An Impossibility Result

We now turn to formalising the preceding claim. In order to simplify the analysis to follow, we focus on a restricted class of distribution samplers. We say that a distribution sampler is well-behaved if it satisfies the following properties:

- It is efficiently sampleable.
- For any i the entropy estimate γ_i of the random variable I_i is fixed, but may vary across different values of i.
- For all $i > 0$ such that $\Pr(\sigma_{i-1}) > 0$ it holds that:

$$\mathrm{H}_\infty\left(I_i \,|\, I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_{q_r}, z_1, \ldots, z_{q_r}, \gamma_1, \ldots, \gamma_{q_r}\right) \geq \gamma_i$$

where $(\sigma_i, I_i, \gamma_i, z_i) = \mathscr{D}(\sigma_{i-1})$ for $i \in \{1, \ldots, q_r\}$ and $\sigma_0 = \varepsilon$.

For any well-behaved distribution sampler $\mathscr{D}$ and any PRNG with input PRNG, let us now consider the experiment of running setup and init to obtain a public paramer pp and an initial state S_0, and then applying a sequence of queries $q_1, \ldots, q_i, \ldots$ where each q_i represents a query to $\mathsf{refresh}$ or next. To any query q_i we associate a tuple $(R_i, S_i, \sigma_i, I_i, \gamma_i)$ that represents the outcome of that query. If q_i is a $\mathsf{refresh}$ query these variables are set by the outputs of $\mathscr{D}$ and $\mathsf{refresh}$, while R_i is set to ε. If q_i is a next query these variables are set to the outputs of next while γ_i is set to zero, I_i is set to the empty string, and $\sigma_i \leftarrow \sigma_{i-1}$. (Note that we deviate slightly here in the notation we use for the output and state of a PRNG with input: we use R_i and S_i to denote *random variables* and we use r_i and s_i respectively to denote *values* assumed by these random variables.)

Now let the function $f : \mathbb{N} \to \mathbb{N}$ where $f(0) = 0$ identify a subsequence $(R_{f(j)}, S_{f(j)}, \sigma_{f(j)}, I_{f(j)}, \gamma_{f(j)})$. We say that a subsequence is *legitimate* if for all $S_{f(j)}$ there exists $f(j-1) \leq c \leq d \leq f(j)$ such that $\sum_c^d \gamma_i \geq \gamma^*$, and all queries between c and d are refresh queries. For ease of notation we let ϵ denote an upper bound on $\mathsf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG}}(\mathscr{A}, \mathscr{D}') + \frac{1}{2^r}$ over all $\mathscr{D}'$ and all $\mathscr{A}$ in some class of adversaries with restricted sources.

With this notation established, we can state the main theorem of this section as follows:

Theorem 5. *For any PRNG with input* PRNG *having associated parameters* (n, l, p)*, any well-behaved distribution sampler* $\mathscr{D}$*, any sequence of queries, any legitimate subsequence identified by the function* f*, any index* j*, and any* $k \in \mathbb{N}$*, it holds that:*

$$\tilde{\mathrm{H}}_\infty \left(\bar{S'}_{f(j)} | R_{f(j)+k}, pp \right) \geq \frac{j+1}{2} \log \left(\frac{1}{\epsilon} \right) - \min(n, l).$$

The proof of the theorem can be found in Appendix D.

This theorem deserves some interpretation. On the left-hand-side, $R_{f(j)+k}$ refers to a particular output received by $\mathscr{B}$ and pp to the public parameters. The theorem says that, conditioned on these, the vector of states $\bar{S'}_{f(j)}$ still has large average min-entropy, provided j is sufficiently large. This is because, on the right-hand-side, $\min(n, l)$ is fixed for a given generator, ϵ is small (so $\log\left(\frac{1}{\epsilon}\right)$ is large), and the first term scales linearly with j, thus attaining arbitrarily large values as j increases. This means that it is impossible for $\mathscr{B}$ to compute or guess the state vector with a good success probability. In short, no adversary, irrespective of its computational resources or backdoor information, can recover all the state information represented by the vector $\bar{S'}_{f(j)}$. In addition the result extends easily to the stronger setting where the adversary is given any sequence of outputs following $R_{f(j)}$, since these will depend only on $S_{f(j)}$ and independently sampled future I values. In that case, we simply replace the $R_{f(j)+k}$ term by any sequence of ouputs following $R_{f(j)}$ and $\min(n, l)$ by n.

5.2 Discussion and Open Problems

Theorem 5 concerns *state* recovery attacks against robust PRNGs with input. It seems plausible to us that the result can be strengthened to say something about the impossibility of recovering old outputs, instead of old states. Likewise, the theorem only concerns the impossibility of recovering *old* states from current outputs, but nothing about the hardness of recovering *future* states or outputs (after refreshing) from current outputs. Informally, the strength of the robustness security notion seems to make such a result plausible, since it essentially requires that a PRNG with input cannot ignore its entropy inputs when refreshing. However, we have not yet proved a formal result in this direction. These are problems that we intend to study in our immediate future work. They relate closely to the kind of impossibility result that would be useful in demonstrating the absence of the kind of effective backdooring that $\mathscr{B}$ might prefer to perform.

This result can also be seen as saying that a PRNG with input is, to some extent, intrinsically immunised against backdooring attacks, since $\mathscr{B}$ cannot recover *all* old states once sufficient entropy has been accumulated in the generator. Here the immunisation is a direct consequence of the nature of the primitive. By contrast, for PRGs, the results of [14] concerning immunisation of PRGs require intrusive changes to the PRG, essentially post-processing the generator's output with either a keyed primitive (a PRF) or a hash with relatively strong security (a random oracle or a Universal Computational Extractor). Moreover, our strengthening of the result of [14], via constructions of forward-secure PRGs that are backdoored in the strong first sense, shows that PRGs cannot resist backdooring in general. So some form of external immunisation is inevitable if PRGs are to resist backdooring.

On the other hand, exploring immunisation for PRNGs with input would still be useful, since, as our constructions in Section 4 show, it is possible to achieve meaningful levels of backdooring for PRNGs with input. Naively, the immunisation techniques of [14] should work equally well for PRNGs with input as they do for PRGs, since a PRNG with input certainly contains within it an implicit PRG, and if that simpler component is immunised, then so should be the more complex PRNG primitive. Furthermore, it may be that PRNGs with input, being informally *harder* to backdoor, could be immunised by applying less intrusive or less idealised cryptographic techniques.

Acknowledgments

Degabriele and Paterson were supported by EPSRC grant EP/M013472/1 (UK Quantum Technology Hub for Quantum Communications Technologies). Schuldt was supported by JSPS KAKENHI Grant Number 15K16006. Woodage was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1)

References

1. Paolo Abeni, Luciano Bello, and Maximiliano Bertacchini. Exploiting DSA-1571: How to break PFS in SSL with EDH, July 2008.
2. Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. Cryptology ePrint Archive, Report 2015/517, 2015. `http://eprint.iacr.org/2015/517`.
3. Thomas Baignères, Cécile Delerablée, Matthieu Finiasz, Louis Goubin, Tancrède Lepoint, and Matthieu Rivain. Trap me if you can – million dollar curve. *IACR Cryptology ePrint Archive*, 2015:1249, 2015.
4. Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05*, pages 203–212, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press.
5. Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 535–552, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.
6. Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.
7. Daniel J. Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko van Someren. Factoring RSA keys from certified smart cards: Coppersmith in the wild. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 341–360, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany.
8. Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, and Christine van Vredendaal. How to manipulate curve standards: a white paper for the black hat. Cryptology ePrint Archive, Report 2014/571, 2014. `http://eprint.iacr.org/2014/571`.
9. Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Sadeghi et al. [30], pages 967–980.
10. Daniel R. L. Brown. A weak-randomizer attack on RSA-OAEP with e = 3. Cryptology ePrint Archive, Report 2005/189, 2005. `http://eprint.iacr.org/2005/189`.
11. Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual EC in TLS implementations. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 319–335. USENIX Association, 2014.
12. Mario Cornejo and Sylvain Ruhault. Characterization of real-life PRNGs under partial state corruption. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 1004–1015, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.
13. Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 579–598, Istanbul, Turkey, March 8–11, 2015. Springer, Heidelberg, Germany.

14. Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 101–126, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.
15. Yevgeniy Dodis, Shien Jin Ong, Manoj Prabhakaran, and Amit Sahai. On the (im)possibility of cryptography with imperfect randomness. In *45th FOCS*, pages 196–205, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.
16. Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergnaud, and Daniel Wichs. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In Sadeghi et al. [30], pages 647–658.
17. Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.
18. Yevgeniy Dodis, Adi Shamir, Noah Stephens-Davidowitz, and Daniel Wichs. How to eat your entropy and have it too - optimal recovery strategies for compromised RNGs. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 37–54, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.
19. Ian Goldberg and David Wagner. Randomness and the Netscape browser. *Dr Dobb's Journal-Software Tools for the Professional Programmer*, 21.1:66–71, 1996.
20. Brett Hemenway, Benoît Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 70–88, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.
21. Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 205–220. USENIX Association, 2012.
22. Eike Kiltz, Adam O'Neill, and Adam Smith. Instantiability of RSA-OAEP under chosen-plaintext attack. Cryptology ePrint Archive, Report 2011/559, 2011. `http://eprint.iacr.org/2011/559`.
23. Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Public keys. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 626–642, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.
24. Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 657–686, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.
25. Bodo Möller. A public-key encryption scheme with pseudo-random ciphertexts. In Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva, editors, *ESORICS 2004*, volume 3193 of *LNCS*, pages 335–351, Sophia Antipolis, French Riviera, France, September 13–15, 2004. Springer, Heidelberg, Germany.
26. Markus Mueller. Debian OpenSSL predictable PRNG bruteforce SSH exploit, May 2008.
27. Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 187–196, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

28. Thomas Ristenpart and Scott Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, 2010.
29. Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. Cryptology ePrint Archive, Report 2015/695, 2015. `http://eprint.iacr.org/2015/695`.
30. Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors. *ACM CCS 13*, Berlin, Germany, November 4–8, 2013. ACM Press.
31. Dan Shumow and Nils Ferguson. On the possibility of a back door in the NIST SP800-90 Dual EC PRNG. Presentation at rump session of CRYPTO 2007, 2007.
32. Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In David Chaum, editor, *CRYPTO'83*, pages 51–67, Santa Barbara, CA, USA, 1983. Plenum Press, New York, USA.
33. Umesh V. Vazirani and Vijay V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 23–30. IEEE Computer Society, 1983.
34. Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In Anja Feldmann and Laurent Mathy, editors, *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference, IMC 2009, Chicago, Illinois, USA, November 4-6, 2009*, pages 15–27. ACM, 2009.
35. Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 62–74, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.
36. Adam Young and Moti Yung. Relationships between Diffie-Hellman and "index oracles". In Carlo Blundo and Stelvio Cimato, editors, *SCN 04*, volume 3352 of *LNCS*, pages 16–32, Amalfi, Italy, September 8–10, 2005. Springer, Heidelberg, Germany.

A Preliminaries from Section 2

A.1 Entropy from Section 2.2

We recall some definitions and results on min-entropy that we will need.

Definition 23. *Let X be a distribution over a set S. The min-entropy of X is defined to be*

$$\mathrm{H}_\infty(X) := -\log(\max_x \Pr[X = x]).$$

Definition 24. *Let X, Z be distributions over a set S. The worst-case min-entropy of X conditioned on Z is defined to be:*

$$\mathrm{H}_\infty(X|Z) := -\log(\max_{x,z} \Pr[X = x \mid Z = z]).$$

The chain rule for worst-case min-entropy is given by:

$$\mathrm{H}_\infty(X, Z) - \mathrm{H}_\infty(Z) \geq \mathrm{H}_\infty(X|Z) \geq \mathrm{H}_\infty(X, Z) - \log(|supp(Z)|).$$

Definition 25. *Let X and Z be distributions over a set S. The average-case guessing probability of X conditioned on Z is defined to be*

$$\mathrm{GP}(X|Z) := \mathbb{E}_{z \leftarrow Z}\left[\max_x \Pr[X = x \mid Z = z]\right].$$

The average-case min-entropy of X conditioned Z is defined to be:

$$\tilde{\mathrm{H}}_\infty(X|Z) := -\log(\mathrm{GP}(X|Z)).$$

Lemma 2. *[17] Let X, Y and Z be distributions over a set S. Then*

$$\tilde{\mathrm{H}}_\infty(X|Y,Z) \geq \tilde{\mathrm{H}}_\infty(X,Y|Z) - \log(|supp(Y)|) \geq \tilde{\mathrm{H}}_\infty(X|Z) - \log(|supp(Y)|),$$

and

$$\tilde{\mathrm{H}}_\infty(X|Y) \geq \mathrm{H}_\infty(X,Y) - \log(|supp(Y)|) \geq \mathrm{H}_\infty(X) - \log(|supp(Y)|).$$

Definition 26. *Let X be a distribution over a set S. The collision-entropy of X is defined to be*

$$\mathrm{H}_2(X) := -\log(\sum_{x \in S} \Pr[X = x]^2).$$

Definition 27. *Let X, Z be distributions where X takes values in the set S. The conditional collision probability of X given Z is defined to be:*

$$\mathrm{CP}(X|Z) := \mathbb{E}_{z \leftarrow Z}\left[\sum_{x \in S} \Pr[X = x \mid Z = z]^2\right].$$

The conditional collision entropy of X conditioned on Z is defined to be:

$$\tilde{\mathrm{H}}_2(X|Z) := -\log(\mathrm{CP}(X|Z)).$$

Definition 28. *Let X and Z be distributions over a set S. Then the statistical distance of X and Z, denoted $\Delta(X,Z)$ is defined to be:*

$$\Delta(X,Z) = \frac{1}{2}\sum_{s \in S} |\Pr[X = s] - \Pr[Z = s]|.$$

Definition 29. *A function $\mathsf{Ext} : \{0,1\}^u \times \{0,1\}^v \to \{0,1\}^w$ is said to be a (k,ϵ)-extractor if for all distributions X over $\{0,1\}^u$ such that $\mathrm{H}_\infty(X) \geq k$, it holds that*

$$\Delta((\mathsf{Ext}(X;A),A),(\mathcal{U}_w,A)) \leq \epsilon$$

where A is uniform over $\{0,1\}^v$.

A.2 Cryptographic Primitives from Section 2.3

Public Key Encryption Schemes

Definition 30. *A public-key encryption (PKE) scheme is a triple of PPT algorithms* $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *where:*

- KGen *takes as input some random coins and returns a public/secret key pair* $(pk, sk) \in \{0,1\}^* \times \{0,1\}^*$*.*
- $\mathsf{Enc} : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *takes as input a public key* pk*, a message* M*, and coins* R*, to return a ciphertext* $C = \mathsf{Enc}(pk, M; R)$*. For ease of notation we will often ommit the coins* R*.*
- $\mathsf{Dec} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *takes as input a ciphertext* C *and secret key* sk*, and returns* $\mathsf{Dec}(sk, C) = M' \in \{0,1\}^* \cup \{\bot\}$*.*

A PKE scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is required to be (perfectly) correct, that is, for all* $(pk, sk) \twoheadleftarrow \mathsf{KGen}$ *and all messages* M*, it hold that:*

$$\Pr\left[\mathsf{Dec}(sk, (\mathsf{Enc}(pk, M)) = M\right] = 1.$$

We further require that encryption be length-regular, meaning that for all $(pk, sk) \twoheadleftarrow \mathsf{KGen}$ *and all message pairs* (M_0, M_1) *such that* $|M_0| = |M_1|$*, it hold that:*

$$\Pr\left[|\mathsf{Enc}(pk, M_0)| = |\mathsf{Enc}(pk, M_1)|\right] = 1.$$

Trapdoor Permutations, and Lossy Trapdoor Permutations

Definition 31. *An* (n, t, δ)*-family of trapdoor one-way permutations is a tuple of PPT algorithms* $\mathsf{TDP} = (\mathsf{G}, \mathsf{S}, \mathsf{F}, \mathsf{F}^{-1})$ *defined as follows:*

- $\mathsf{G} : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ *takes random coins as input, and outputs a public/secret-key pair* $(\mathsf{PK}, \mathsf{SK})$*. Each* PK *implicitly defines a permutation over the domain* D_{PK}*.*
- S *takes* PK *and random coins as input, and outputs an element* $x \in \mathsf{D}_{\mathsf{PK}}$*; where* $x \twoheadleftarrow \mathsf{S}(\mathsf{PK})$ *is identically distributed to* $x \twoheadleftarrow \mathsf{D}_{\mathsf{PK}}$*. We require that for any* PK *output by* G*, it holds that* $2^{n-1} \leq |\mathsf{D}_{\mathsf{PK}}| \leq 2^n$*.*
- $\mathsf{F} : \{0,1\}^* \times \mathsf{D}_{\mathsf{PK}} \to \mathsf{D}_{\mathsf{PK}}$ *takes as input* PK *and* $x \in \mathsf{D}_{\mathsf{PK}}$*, and outputs* $y \in \mathsf{D}_{\mathsf{PK}}$*. We write this as* $y \leftarrow \mathsf{F}_{\mathsf{PK}}(x)$*.*
- F^{-1} *takes as input* SK *and* $y \in \mathsf{D}_{\mathsf{PK}}$ *and outputs* $z \in \mathsf{D}_{\mathsf{PK}}$ *such that* $\mathsf{F}_{\mathsf{PK}}(z) = y$*. We write this as* $z \leftarrow \mathsf{F}^{-1}_{\mathsf{PK}}(y)$*.*
- *For all adversaries* $\mathscr{A}$ *running in time* t*, it holds that*

$$\mathsf{Adv}^{inv}_{\mathsf{TDP}}(\mathscr{A}) := \Pr\left[\text{TDP-INV}^{\mathscr{A}}_{\mathsf{TDP}} \Rightarrow \mathsf{true}\right] \leq \delta$$

where the game $\text{TDP-INV}^{\mathscr{A}}_{\mathsf{TDP}}$ *is defined in Figure 12.*

We next introduce lossy trapdoor permutations, following [27].

```
Game $\text{TDP-INV}^{\mathscr{A}}_{\mathsf{TDP}}$
$(\mathsf{PK}, \mathsf{SK}) \twoheadleftarrow \mathsf{G}()$
$x \twoheadleftarrow \mathsf{S}(\mathsf{PK})$
$y \leftarrow \mathsf{F}_{\mathsf{PK}}(x)$
$z \leftarrow \mathscr{A}(\mathsf{PK}, y)$
return $(z = x)$
```

```
Game $\text{LTDP-DIST}^{\mathscr{A}}_{\mathsf{LTDP}}$
$b \twoheadleftarrow \{0, 1\}$
$(\mathsf{PK}, \mathsf{SK}) \twoheadleftarrow \mathsf{G}_b()$
$b' \leftarrow \mathscr{A}(\mathsf{PK})$
return $(b = b')$
```

Fig. 12: The games $\text{TDP-INV}^{\mathscr{A}}_{\mathsf{TDP}}$ and $\text{LTDP-DIST}^{\mathscr{A}}_{\mathsf{LTDP}}$.

Definition 32. *A family of (n, k, t, ϵ)-lossy trapdoor one-way permutations is a tuple of PPT algorithms $\mathsf{LTDP} = (\mathsf{G}_0, \mathsf{G}_1, \mathsf{S}, \mathsf{F}, \mathsf{F}^{-1})$ with functionality as follows:*

- G_0 *takes random coins as input and outputs* $(\mathsf{PK}, \perp)$. *Each* PK *implicitly defines a function* F_{PK} *over domain* D_{PK} *whose image has size at most* $2^{-k}|\mathsf{D}_{\mathsf{PK}}|$.
- G_1 *takes random coins as input and outputs* $(\mathsf{PK}, \mathsf{SK})$. *Each* PK *implicitly defines a permutation* F_{PK} *over* D_{PK}, *with inverse* $\mathsf{F}^{-1}_{\mathsf{SK}}$.
- S *takes* PK *and random coins as input, and outputs an element* $x \in \mathsf{D}_{\mathsf{PK}}$; *where* $x \twoheadleftarrow \mathsf{S}(\mathsf{PK})$ *is identically distributed to* $x \twoheadleftarrow \mathsf{D}_{\mathsf{PK}}$. *We require that for any* PK *output by* G_0 *or* G_1, *it holds that* $2^{n-1} \leq |\mathsf{D}_{\mathsf{PK}}| \leq 2^n$.
- *For all adversaries* $\mathscr{A}$ *running in time* t *it holds that*

$$\mathsf{Adv}^{\text{dist}}_{\mathsf{LTDP}}(\mathscr{A}) := 2|\Pr\left[\text{LTDP-DIST}^{\mathscr{A}}_{\mathsf{LTDP}} \Rightarrow \mathsf{true}\right] - \frac{1}{2}| \leq \epsilon$$

where game $\text{LTDP-DIST}^{\mathscr{A}}_{\mathsf{LTDP}}$ *is defined in Figure 12.*

As shown in [27], the indistinguishability of real and lossy keys immediately implies that $(\mathsf{G}_1, \mathsf{S}, \mathsf{F}, \mathsf{F}^{-1})$ is an $(n, t, \epsilon + 2^{-k})$-family of trapdoor one-way permutations.

Kiltz et al. show in [22] that RSA is a lossy trapdoor one-way permutation under the Φ-hiding assumption.

A.3 Proof of Theorem 1

Proof. The result follows from a straightforward adaptation of the proof in [16] to handle our changes to the formalism; for completeness, we provide a sketch proof here.

Let $(\mathscr{A}, \mathscr{D})$ be an attacker/sampler pair in Game $\text{ROB}^{\mathscr{D},\mathscr{A}}_{\mathsf{PRNG},\gamma^*}$ against PRNG. We shall construct a hybrid argument based upon the attacker's Ror queries, where we assume $\mathscr{A}$ makes q_n such queries. We say that a Ror query is uncompromised if $c \geq \gamma^*$ at the point of the query. We further divide uncompromised Ror queries into those which are *preserving* and those which are *recovering*. We say that a Ror query is preserving if $c \geq \gamma^*$ throughout the period between the previous Ror query and the current one. We say a Ror query is *recovering* if $c = 0$ at some point between the previous Ror query and the current one. For a recovering Ror query, we call the most recent query for which $c \leftarrow 0$ the most recent entropy drain (mRED) query.

We define a series of hybrid games as follows. Let $\mathsf{Game}\ 0$ be the real-or-random ROB game as defined in Figure 3. We now define a series of modified games, where for the first i Ror queries in $\mathsf{Game}\ i$, if that query is uncompromised, then the challenger returns $r \twoheadleftarrow \{0,1\}^l$ and sets the state of the PRNG to $s \twoheadleftarrow \mathsf{init}(pp)$, regardless of the challenge bit. We also define an intermediate Game $(i + \frac{1}{2})$, defined such that if the $(i+1)^{\text{st}}$ Ror query is preserving then the challenger acts as in Game $i+1$, whereas if the $(i+1)^{\text{st}}$ Ror query is recovering the challenger acts as in Game i.

We claim that for $i \in \{0, \ldots, q_n - 1\}$, $\mathsf{Game}\ i$ is indistinguishable form $\mathsf{Game}\ i + 1/2$, and that $\mathsf{Game}\ i + 1/2$ is indistinguishable from $\mathsf{Game}\ i + 1$.

Claim. If the PRNG PRNG has (t, ϵ_{pre})-PRE security, then for any attacker/distinguisher pair $(\mathscr{A}, \mathscr{D})$ running in time $t' \approx t$, it holds that

$$|\Pr[\mathsf{Game}\ i \Rightarrow \mathsf{true}] - \Pr[\mathsf{Game}\ i + 1/2 \Rightarrow \mathsf{true}]| \leq \epsilon_{pre}.$$

Claim. If the PRNG PRNG has $(t, q_r, \gamma^*, \epsilon_{rec})$-REC security, then for any attacker/distinguisher pair $(\mathscr{A}, \mathscr{D})$ running in time $t' \approx t$, it holds that

$$|\Pr[\mathsf{Game}\ i + 1/2 \Rightarrow \mathsf{true}] - \Pr[\mathsf{Game}\ i + 1 \Rightarrow \mathsf{true}]| \leq \epsilon_{rec}.$$

The proof of the first claim utilises the fact that an attacker $\mathscr{A}_1$ in Game $\mathrm{PRE}^{\mathscr{A}}_{\mathsf{PRNG}}$ can simulate Game $\mathrm{ROB}^{\mathscr{D},\mathscr{A}}_{\mathsf{PRNG},\gamma^*}$ for $\mathscr{A}$ using the code of the sampler $\mathscr{D}$, and substitute his challenge query in the place of any preserving Ror query. Likewise, for the latter, an attacker/sampler pair ($\mathscr{A}_2$, $\mathscr{D}$) in Game $\mathrm{REC}^{\mathscr{D},\mathscr{A},q_r}_{\mathsf{PRNG},\gamma^*}$ can simulate Game $\mathrm{ROB}^{\mathscr{D},\mathscr{A}}_{\mathsf{PRNG},\gamma^*}$ for $\mathscr{A}$ using the Sam oracle and the unused inputs received at the conclusion of the game. $\mathscr{A}_2$ can then substitute his challenge value in the place of any recovering Ror query by setting the initial state in Game $\mathrm{REC}^{\mathscr{D},\mathscr{A},q_r}_{\mathsf{PRNG},\gamma^*}$ equal to the state immediately following the corresponding mRED query in the simulated Game $\mathrm{ROB}^{\mathscr{D},\mathscr{A}}_{\mathsf{PRNG},\gamma^*}$.

Combining the above, we get

$$|\Pr[\mathsf{Game}\ 0 \Rightarrow \mathsf{true}] - \Pr[\mathsf{Game}\ q_n \Rightarrow \mathsf{true}]| \leq q_n(\epsilon_{pre} + \epsilon_{rec}).$$

Notice that in $\mathsf{Game}\ q_n$, the challenger responds to each uncompromised Ror query by returning $r \twoheadleftarrow \{0,1\}^l$ and setting $s \twoheadleftarrow \mathsf{init}(pp)$. Therefore the game is independent of the challenge bit b, implying $\Pr[\mathsf{Game}\ q_n \Rightarrow \mathsf{true}] = \frac{1}{2}$.

We conclude that

$$\begin{aligned}\mathsf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG}}(\mathscr{A},\mathscr{D}) &= |\Pr[\mathsf{Game}\ 0 \Rightarrow \mathsf{true}] - \frac{1}{2}| \\ &= |\Pr[\mathsf{Game}\ 0 \Rightarrow \mathsf{true}] - \Pr[\mathsf{Game}\ q_n \Rightarrow \mathsf{true}]| \\ &\leq q_n(\epsilon_{pre} + \epsilon_{rec})\end{aligned}$$

as required.

B Proofs for Section 3

B.1 Proof of Theorem 2

Proof. The correctness of $\mathcal{E}$ immediately implies that $\mathsf{Adv}^{\mathrm{first}}_{\mathsf{PRG}}(\mathscr{B},q,i) = 1$. More difficult is to prove the security of $\mathsf{PRG} = (\mathsf{setup},\mathsf{init},\mathsf{next})$ against a standard PRG adversary $\mathscr{A}$ in the forward-security game $\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}}$. This follows from the following sequence of inequalities, with the corresponding security games $\mathcal{G}^{\mathscr{A}}_0,\ldots,\mathcal{G}^{\mathscr{A}}_6$ being shown in Figure 13 and Figure 14, (and where the code in the procedure $\mathsf{challenge}$ in each game encapsulates how the security game computes values that are provided to the adversary):

$$\begin{aligned}\mathsf{Adv}^{\mathrm{fwd}}_{\mathsf{PRG}}(\mathscr{A},q) &= 2|\Pr\left[\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}} \Rightarrow \mathsf{true}\right] - \frac{1}{2}| \\ &= |\Pr\left[\mathcal{G}^{\mathscr{A}}_0 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (1)\\ &= |\Pr\left[\mathcal{G}^{\mathscr{A}}_1 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (2)\\ &= |\Pr\left[\mathcal{G}^{\mathscr{A}}_2 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (3)\\ &= \mathsf{Adv}^{\mathrm{ind\$-cpa}}_{\mathcal{E}}(\mathscr{A}_1) + |\Pr\left[\mathcal{G}^{\mathscr{A}}_3 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (4)\\ &\leq \delta + |\Pr\left[\mathcal{G}^{\mathscr{A}}_3 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (5)\\ &\leq \delta + \Pr[\mathsf{bad}_1] + |\Pr\left[\mathcal{G}^{\mathscr{A}}_4 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (6)\\ &= \delta + \mathsf{Adv}^{\mathrm{inv}}_{\mathsf{TDP}}(\mathscr{A}_2) + |\Pr\left[\mathcal{G}^{\mathscr{A}}_4 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (7)\\ &\leq \delta + \epsilon + 2^{-k} + |\Pr\left[\mathcal{G}^{\mathscr{A}}_4 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (8)\\ &\leq \delta + \epsilon + 2^{-k} + \Pr[\mathsf{bad}_2 \text{ in } \mathcal{G}_4] \\ &\qquad + |\Pr\left[\mathcal{G}^{\mathscr{A}}_5 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (9)\\ &\leq 2\delta + 3\epsilon + 2^{-(k-1)} + \Pr[\mathsf{bad}_2 \text{ in } \mathcal{G}^*_2] \\ &\qquad + |\Pr\left[\mathcal{G}^{\mathscr{A}}_5 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (10)\end{aligned}$$

$$\begin{aligned}&\leq 2\delta + 3\epsilon + 2^{-(k-1)} + q \cdot 2^{-(k-1)} \\ &\qquad + |\Pr\left[\mathcal{G}^{\mathscr{A}}_5 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right]| && (11)\\ &= 2\delta + 3\epsilon + (q+1)2^{-(k-1)}. && (12)\end{aligned}$$

We begin by observing that Game $\mathcal{G}_0^{\mathscr{A}}$ is identical to the Game PRG-FWD$_{\mathsf{PRG}}^{\mathscr{A},q}$ with challenge bit $b=1$, and that $\mathcal{G}_6^{\mathscr{A}}$ is identical to the Game PRG-FWD$_{\mathsf{PRG}}^{\mathscr{A},q}$ with challenge bit $b=0$, justifying (1).

Game $\mathcal{G}_1^{\mathscr{A}}$ is identical to $\mathcal{G}_0^{\mathscr{A}}$ except we change the way in which the random oracle RO responds to queries on inputs M_j for $j=1,\ldots,q$. Since these inputs are formally as yet undefined in both games (they are first defined in $\mathcal{G}_3^{\mathscr{A}}$), this is a purely syntactic change and the two games run identically. It follows that $\Pr\left[\mathcal{G}_0^{\mathscr{A}} \Rightarrow 1\right] = \Pr\left[\mathcal{G}_1^{\mathscr{A}} \Rightarrow 1\right]$, justifying (2).

Game $\mathcal{G}_2^{\mathscr{A}}$ is identical to $\mathcal{G}_1^{\mathscr{A}}$ except we change the way in which the final state s_q is computed. In the former, $s_q \leftarrow \mathsf{F}_{\mathsf{PK}}^{q}(s_0)$, where $s_0 \twoheadleftarrow \mathsf{S}(\mathsf{PK})$, whereas in the latter, $s_q \leftarrow \mathsf{F}_{\mathsf{PK}}(s_0)$, where $s_0 \twoheadleftarrow \mathsf{S}(\mathsf{PK})$. Since applying a permutation to a randomly sampled element from its domain is equivalent to randomly sampling an element from its range, this implies that s_q is identically distributed in the two games, and so $\Pr\left[\mathcal{G}_1^{\mathscr{A}} \Rightarrow 1\right] = \Pr\left[\mathcal{G}_2^{\mathscr{A}} \Rightarrow 1\right]$, justifying (3).

At this point, we also define $\mathcal{G}_{2^*}^{\mathscr{A}}$, which lies perpendicular to $\mathcal{G}_2^{\mathscr{A}}$, although we shall not utilise it until later in the proof when it is used to bound the probability of a flag being set. $\mathcal{G}_{2^*}^{\mathscr{A}}$ is identical to $\mathcal{G}_2^{\mathscr{A}}$, except now the challenger picks a lossy, rather than injective, key $\mathsf{PK}' \twoheadleftarrow \mathsf{G}_0$. Since both games are perfectly simulatable by an adversary $\mathscr{A}_3$ in the LTDP distinguishing game, it follows that

$$|\Pr\left[\mathcal{G}_2^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_{2^*}^{\mathscr{A}} \Rightarrow 1\right]| \leq \mathsf{Adv}_{\mathsf{LTDP}}^{\mathsf{dist}}(\mathscr{A}_3) \leq \epsilon. \tag{13}$$

We now return to the previous line of argument. We define game $\mathcal{G}_3^{\mathscr{A}}$, which is identical to $\mathcal{G}_2^{\mathscr{A}}$, except we replace the randomly sampled PRG outputs $r_1,\ldots,r_q \twoheadleftarrow (\{0,1\}^q)^l$, with pseudorandom encryptions of pre-images of s_0. Now we define $r_j \leftarrow \mathsf{Enc}(pk, M_j; R_j)$, where $M_j \leftarrow \mathsf{F}_{\mathsf{PK}}^{-q+j}(s_0)$ and $R_j \twoheadleftarrow \mathsf{Coins}(\mathcal{E})$ for $j = 1,\ldots k$. We also set a flag bad_1, but this does not affect the outcome of the game. These games are perfectly simulatable by an IND\$-CPA adversary $\mathscr{A}_1$ against $\mathcal{E}$ (who, in particular, generates $(\mathsf{PK},\mathsf{SK}) \twoheadleftarrow \mathsf{G}_0$, and so can efficiently compute the required preimages), implying that

$$|\Pr\left[\mathcal{G}_3^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_2^{\mathscr{A}} \Rightarrow 1\right]| \leq \mathsf{Adv}_{\mathcal{E}}^{\mathrm{ind\$-cpa}}(\mathscr{A}_1) \leq \delta,$$

and justifying (4), (5).

Game $\mathcal{G}_4^{\mathscr{A}}$, is identical to $\mathcal{G}_3^{\mathscr{A}}$, except we change the way we sample the randomness for the encryptions. Rather than choosing $R_j \twoheadleftarrow \mathsf{Coins}(\mathcal{E})$ for $j = 1,\ldots k$ as in $\mathcal{G}_3^{\mathscr{A}}$, we now use $a_j \leftarrow \mathsf{RO}(M_j)$, $j=1,\ldots,k$ where M_j is the message to be encrypted. These games run identically unless the flag bad_1 is set, in which case we have found distinct $1 \leq k, l \leq q$ such that $M_k = M_l$, leading to outputs $r_k = r_l$ in $\mathcal{G}_4^{\mathscr{A}}$, justifying (6). The occurrence of bad_1 implies the existence of an adversary $\mathscr{A}_2$ in game TDP-INV$_{\mathsf{TDP}}^{\mathscr{A}_2}$ who can use this to invert the trapdoor one-way permutation with probability 1. Therefore $\Pr[\mathsf{bad}_1] \leq \mathsf{Adv}_{\mathsf{TDP}}^{\mathrm{inv}}(\mathscr{A}_2) \leq \epsilon + 2^{-k}$, implying (6), (7) and (8).

Now $\mathcal{G}_5^{\mathscr{A}}$ is the same as $\mathcal{G}_4^{\mathscr{A}}$ except we change the way in which the random oracle RO responds to queries. Whereas in $\mathcal{G}_4^{\mathscr{A}}$, RO responded to each of

$\mathscr{A}'s$ fresh queries with a random string (effectively 'forgetting' queries during the computation stage), now RO 'remembers' these values, and will respond to queries on target input M_j to return the corresponding value a_j which was used in the encryptions. Notice that these two games run identically unless the flag bad_2 is set, and the probability that this event occurs is identical in both games, so the Fundamental Lemma of Game Playing justifies (9). We now bound the probability $\Pr[\mathsf{bad}_2 \text{ is set in } \mathcal{G}_4]$. Collecting what we have so far, a straightforward reduction implies that

$$\Pr[\mathsf{bad}_2 \text{ is set in } \mathcal{G}_4] \leq \Pr[\mathsf{bad}_2 \text{ is set in } \mathcal{G}_2^*] + \delta + 2\epsilon + 2^{-k},$$

justifying (10). Now for each $j = 1, \ldots, q$, the maximum probability that $\mathscr{A}$ guesses $\mathsf{F}_{\mathsf{PK}}^{-q+j}(s_0)$ given PK' and $\mathsf{F}_{\mathsf{PK}'}(s_0)$ is

$$\begin{aligned}
\mathrm{GP}(\mathsf{F}_{\mathsf{PK}}^{-q+j}(s_0)|\mathsf{PK}', \mathsf{F}_{\mathsf{PK}'}(s_0)) &= 2^{-\,\tilde{\mathrm{H}}_\infty\left(\mathsf{F}_{\mathsf{PK}}^{-q+j}(s_0)|\mathsf{PK}',\mathsf{F}_{\mathsf{PK}'}(s_0)\right)} \\
&\leq 2^{-(\tilde{\mathrm{H}}_\infty\left(-\mathsf{F}_{\mathsf{PK}}^{-q+j}(s_0)|\mathsf{PK}'\right)-\log(|supp(\mathsf{F}_{\mathsf{PK}'}(s_0))|))} \\
&\leq 2^{-(\tilde{\mathrm{H}}_\infty\left(-\mathsf{F}_{\mathsf{PK}}^{-q+j}(s_0)|\mathsf{PK}'\right)-(n-k))} \\
&\leq 2^{-((n-1)-(n-k))} \\
&\leq 2^{-(k-1)}.
\end{aligned}$$

The first equality follows from Definition 25. The second inequality follows from an application of Lemma 2. The third inequality follows from the fact that, by the definition of an (n, k, t, ϵ)-family of lossy trapdoor permutations, $\mathsf{F}_{\mathsf{PK}'}(s_0)$ can take at most $2^{-k}|\mathsf{D}_{\mathsf{PK}'}|$ possible values where $|\mathsf{D}_{\mathsf{PK}'}| \leq 2^n$. The fourth inequality follows since by definition PK (generated independently from PK') gives a permutation and $s_0 \twoheadleftarrow \mathsf{S}(\mathsf{PK})$ is equivalent to $s_0 \twoheadleftarrow \mathsf{D}_{\mathsf{PK}}$ where $|\mathsf{D}_{\mathsf{PK}}| \geq 2^{n-1}$, thus implying that $\tilde{\mathrm{H}}_\infty\left(\mathsf{F}_{\mathsf{PK}}^{-q+j}(s_0)|\mathsf{PK}'\right) \geq n-1$ for each $j = 1, \ldots, q$. Finally, from an application of the union bound, we conclude that $\Pr[\mathsf{bad}_2 \text{ is set in } \mathcal{G}_1^*] \leq q \cdot 2^{-(k-1)}$, implying (11).

Finally, we define $\mathcal{G}_6^{\mathscr{A}}$, in which we reverse the change of variable and replace $s_j = \mathsf{F}_{\mathsf{PK}}^{-q+j}(s_0)$ with $s_j = \mathsf{F}_{\mathsf{PK}}^{j}(s_0)$ for $j = 1, \ldots, q$. By the same argument as before, this does not affect the distribution of the two games, so $\Pr\left[\mathcal{G}_5^{\mathscr{A}} \Rightarrow 1\right] = \Pr\left[\mathcal{G}_6^{\mathscr{A}} \Rightarrow 1\right]$, justifying (12). We conclude with the advantage term, and the proof is complete.

```
Game $\mathcal{G}_0, \mathcal{G}_1$
setup
$(pk, sk) \twoheadleftarrow \mathsf{KGen}$
$(\mathsf{PK}, \mathsf{SK}) \twoheadleftarrow \mathsf{G}_1$
$pp \leftarrow (pk, \mathsf{PK})$
$bk \leftarrow (sk, \mathsf{SK})$
return $(pp, bk)$
init$(pp)$
$(pk, \mathsf{PK}) \leftarrow pp$
$s_0 \twoheadleftarrow \mathsf{S}(\mathsf{PK})$
return $(s_0)$
challenge$(pp, s_0)$
$(pk, \mathsf{PK}) \leftarrow pp$
$r_1, \ldots, r_q \twoheadleftarrow (\{0,1\}^q)^l$
$s_q \leftarrow \mathsf{F}^q_{\mathsf{PK}}(s_0)$
$b' \leftarrow \mathscr{A}^{\mathsf{RO}}(pp, r_1,$
        $\ldots, r_q, s_q)$
return $(b' = 1)$
```

```
Game $\mathcal{G}_2, \boxed{\mathcal{G}_2^*}$
setup
$(pk, sk) \twoheadleftarrow \mathsf{KGen}$
$(\mathsf{PK}, \mathsf{SK}) \twoheadleftarrow \mathsf{G}_1$
$pp \leftarrow (pk, \mathsf{PK})$
$bk \leftarrow (sk, \mathsf{SK})$
$\boxed{(\mathsf{PK}', \perp) \twoheadleftarrow \mathsf{G}_0}$
$\boxed{pp \leftarrow (pk, \mathsf{PK}')}$
$\boxed{bk \leftarrow (sk, \perp)}$
return $(pp, bk)$
init$(pp)$
$(pk, \mathsf{PK}) \leftarrow pp$
$s_0 \twoheadleftarrow \mathsf{S}(\mathsf{PK})$
return $(s_0)$
challenge$(pp, s_0)$
$(pk, \mathsf{PK}) \leftarrow pp$
$r_1, \ldots, r_q \twoheadleftarrow (\{0,1\}^q)^l$
$s_q \leftarrow \mathsf{F}_{\mathsf{PK}}(s_0)$
$b' \leftarrow \mathscr{A}^{\mathsf{RO}}(pp, r_1,$
        $\ldots, r_q, s_q)$
return $(b' = 1)$
```

```
Game $\mathcal{G}_3$
setup
$(pk, sk) \twoheadleftarrow \mathsf{KGen}$
$(\mathsf{PK}, \mathsf{SK}) \twoheadleftarrow \mathsf{G}_1$
$pp \leftarrow (pk, \mathsf{PK})$
$bk \leftarrow (sk, \mathsf{SK})$
return $(pp, bk)$
init$(pp)$
$(pk, \mathsf{PK}) \leftarrow pp$
$s_0 \twoheadleftarrow \mathsf{S}(\mathsf{PK})$
return $(s_0)$
challenge$(pp, s_0)$
$(pk, \mathsf{PK}) \leftarrow pp$
for $j = 1, \ldots, q$
  $M_j \leftarrow \mathsf{F}^{-q+j}_{\mathsf{PK}}(s_0)$
    if $M_k = M_l \wedge k \neq l$
      $\mathsf{bad}_1 \leftarrow \mathsf{true}$
  $R_j \twoheadleftarrow \mathsf{Coins}(\mathcal{E})$
  $r_j \leftarrow \mathsf{Enc}(pk, M_j; R_j)$
$s_q \leftarrow \mathsf{F}_{\mathsf{PK}}(s_0)$
$b' \leftarrow \mathscr{A}^{\mathsf{RO}}(pp, r_1,$
        $\ldots, r_q, s_q)$
return $(b' = 1)$
```

```
On Query $\mathsf{RO}(M)$ in $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_{2^*}, \mathcal{G}_3,$
if $M = M_j$ for $j = 1, \ldots, q$
  then $\mathsf{bad}_2 \leftarrow \mathsf{true}$
return $a_j \twoheadleftarrow \mathsf{Coins}(\mathcal{E})$
```

```
On Query $\mathsf{RO}(M)$ in $\mathcal{G}_0$
if $M = M_j$ for $j = 1, \ldots, q$
  then $\mathsf{bad}_2 \leftarrow \mathsf{true}$
 return $a_j$
else return $a_j \twoheadleftarrow \mathsf{Coins}(\mathcal{E})$
```

Fig. 13: Games $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_{2^*}, \mathcal{G}_3$ for proof of Theorem 2.

```
Game $\mathcal{G}_4, \mathcal{G}_5$
setup
$(pk, sk) \twoheadleftarrow \mathsf{KGen}$
$(\mathsf{PK}, \mathsf{SK}) \twoheadleftarrow \mathsf{G}_1$
$pp \leftarrow (pk, \mathsf{PK})$
$bk \leftarrow (sk, \mathsf{SK})$
return $(pp, bk)$
init$(pp)$
$(pk, \mathsf{PK}) \leftarrow pp$
$s_0 \twoheadleftarrow \mathsf{S}(\mathsf{PK})$
return $(s_0)$
challenge$(pp, s_0)$
$(pk, \mathsf{PK}) \leftarrow pp$
for $j = 1, \ldots, q$
  $M_j \leftarrow \mathsf{F}_{\mathsf{PK}}^{-q+j}(s_0)$
    if $M_k = M_l \wedge k \neq l$
      $\mathsf{bad}_1 \leftarrow \mathsf{true}$
  $a_j \leftarrow \mathsf{RO}(M_j)$
  $r_j \leftarrow \mathsf{Enc}(pk, M_j; a_j)$
$s_q \leftarrow \mathsf{F}_{\mathsf{PK}}(s_0)$
$b' \leftarrow \mathscr{A}^{\mathsf{RO}}(pp, r_1, \ldots, r_q, s_q)$
return $(b' = 1)$
```

```
Game $\mathcal{G}_6$
setup
$(pk, sk) \twoheadleftarrow \mathsf{KGen}$
$(\mathsf{PK}, \mathsf{SK}) \twoheadleftarrow \mathsf{G}_1$
$pp \leftarrow (pk, \mathsf{PK})$
$bk \leftarrow (sk, \mathsf{SK})$
return $(pp, bk)$
init$(pp)$
$(pk, \mathsf{PK}) \leftarrow pp$
$s_0 \twoheadleftarrow \mathsf{S}(\mathsf{PK})$
return $(s_0)$
challenge$(pp, s_0)$
$(pk, \mathsf{PK}) \leftarrow pp$
for $j = 1, \ldots, q$
  $M_j \leftarrow \mathsf{F}_{\mathsf{PK}}^{j}(s_0)$
    if $M_k = M_l \wedge k \neq l$
      $\mathsf{bad}_1 \leftarrow \mathsf{true}$
  $a_j \leftarrow \mathsf{RO}(M_j)$
  $r_j \leftarrow \mathsf{Enc}(pk, M_j; a_j)$
$s_q \leftarrow \mathsf{F}_{\mathsf{PK}}^{q}(s_0)$
$b' \leftarrow \mathscr{A}^{\mathsf{RO}}(pp, r_1, \ldots, r_q, s_q)$
return $(b' = 1)$
```

```
On Query $\mathsf{RO}(M)$ in $\mathcal{G}_4$
if $M = M_j$ for $j = 1, \ldots, q$
  then $\mathsf{bad}_2 \leftarrow \mathsf{true}$
return $a_j \twoheadleftarrow \mathsf{Coins}(\mathcal{E})$
```

```
On Query $\mathsf{RO}(M)$ in $\mathcal{G}_5, \mathcal{G}_6$
if $M = M_j$ for $j = 1, \ldots, q$
  then $\mathsf{bad}_2 \leftarrow \mathsf{true}$
  return $a_j$
else return $a_j \twoheadleftarrow \mathsf{Coins}(\mathcal{E})$
```

Fig. 14: Games $\mathcal{G}_4, \mathcal{G}_5, \mathcal{G}_6$ for proof of Theorem 2.

B.2 Proof of Theorem 3

Proof. The correctness and re-randomisation properties of $\mathcal{E}$ together imply that $\mathsf{Adv}^{\text{first}}_{\mathsf{PRG}}(\mathscr{B}, q, i) = 1$. It remains to prove the FWD-security of $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ against a non-backdoored attacker $\mathscr{A}$ in Game $\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}}$. This follows from the following sequence of inequalities, with the corresponding security games $\mathcal{G}^{\mathscr{A}}_0, \ldots, \mathcal{G}^{\mathscr{A}}_8$ being shown in Figures 15, 16, and 17:

$$
\begin{aligned}
\mathsf{Adv}^{\text{fwd}}_{\mathsf{PRG}}(\mathscr{A}, q) &= 2|\Pr\left[\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}} \Rightarrow \mathsf{true}\right] - \frac{1}{2}| \\
&= |\Pr\left[\mathcal{G}^{\mathscr{A}}_0 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (14) \\
&\leq 2 \cdot \mathsf{Adv}^{\text{ind\$-cpa}}_{\mathcal{E}}(\mathscr{A}_1) + |\Pr\left[\mathcal{G}^{\mathscr{A}}_1 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (15) \\
&\leq 2\delta + |\Pr\left[\mathcal{G}^{\mathscr{A}}_1 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (16) \\
&\leq 2\delta + \mathsf{Adv}^{\text{fwd}}_{\mathsf{PRG}'}(\mathscr{A}_2) + |\Pr\left[\mathcal{G}^{\mathscr{A}}_2 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (17) \\
&\leq 2\delta + \epsilon_{fwd} + |\Pr\left[\mathcal{G}^{\mathscr{A}}_2 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (18) \\
&\leq 2\delta + \epsilon_{fwd} + \mathit{\Delta}(C_0, \mathsf{Rand}(C_0; t'_1, \ldots, t'_q)) \\
&\qquad + |\Pr\left[\mathcal{G}^{\mathscr{A}}_3 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (19) \\
&\leq 2\delta + \epsilon_{fwd} + q\nu + |\Pr\left[\mathcal{G}^{\mathscr{A}}_3 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (20) \\
&\leq 2\delta + \epsilon_{fwd} + q\nu + \mathsf{Adv}^{\text{ind\$-cpa}}_{\mathcal{E}}(\mathscr{A}_3) \\
&\qquad + |\Pr\left[\mathcal{G}^{\mathscr{A}}_4 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (21) \\
&\leq 2\delta + \epsilon_{fwd} + q\nu + \delta + |\Pr\left[\mathcal{G}^{\mathscr{A}}_4 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (22) \\
&\leq 3\delta + \epsilon_{fwd} + q\nu + \mathsf{Adv}^{\text{ind\$-cpa}}_{\mathcal{E}}(\mathscr{A}_4) \\
&\qquad + |\Pr\left[\mathcal{G}^{\mathscr{A}}_5 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (23) \\
&\leq 3\delta + \epsilon_{fwd} + q\nu + \delta + |\Pr\left[\mathcal{G}^{\mathscr{A}}_5 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (24) \\
&\leq 4\delta + \epsilon_{fwd} + q\nu + q(q+1)\nu/2 \\
&\qquad + |\Pr\left[\mathcal{G}^{\mathscr{A}}_6 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (25) \\
&\leq 4\delta + \epsilon_{fwd} + q(q+3)\nu/2 + \mathsf{Adv}^{\text{fwd}}_{\mathsf{PRG}'}(\mathscr{A}_5) \\
&\qquad + |\Pr\left[\mathcal{G}^{\mathscr{A}}_7 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (26) \\
&\leq 4\delta + \epsilon_{fwd} + q(q+3)\nu/2 + \epsilon_{fwd} \\
&\qquad + |\Pr\left[\mathcal{G}^{\mathscr{A}}_7 \Rightarrow 1\right] - \Pr\left[\mathcal{G}^{\mathscr{A}}_8 \Rightarrow 1\right]| && (27) \\
&\leq 4\delta + 2\epsilon_{fwd} + q(q+3)\nu/2 + 2 \cdot \mathsf{Adv}^{\text{ind\$-cpa}}_{\mathcal{E}}(\mathscr{A}_6) && (28) \\
&\leq 4\delta + 2\epsilon_{fwd} + q(q+3)\nu/2 + 2\delta && (29) \\
&= 6\delta + 2\epsilon_{fwd} + q(q+3)\nu/2. && (30)
\end{aligned}
$$

We begin by observing that game $\mathcal{G}_0$ is identical to $\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}}$ with challenge bit $b = 1$, whilst $\mathcal{G}_8$ is identical to game $\text{PRG-FWD}^{\mathscr{A},q}_{\mathsf{PRG}}$ with challenge bit $b = 0$, justifying (14).

Game $\mathcal{G}_1$ is identical to $\mathcal{G}_0$ except that rather than encrypt the initial state $s_0 \twoheadleftarrow \mathsf{init}'(pp')$ of the internal PRG PRG' in the BPRG state S, we encrypt an independently generated state $M \twoheadleftarrow \mathsf{init}'(pp')$. An adversary $\mathscr{A}_1$ in the IND-CPA game against $\mathcal{E}$ can perfectly simulate both of these games (in particular, $\mathscr{A}_1$ generates the initial PRG state $s_0 \twoheadleftarrow \mathsf{init}'(pp')$ himself, and so can compute the required PRG output), justifying (15), (16).

Game $\mathcal{G}_2$ is identical to $\mathcal{G}_1$ except that we now use truly random strings, as opposed to PRG outputs, to re-randomize C_0. An adversary $\mathscr{A}_2$ in the forward-security game against PRG' with challenge bit b' can perfectly simulate both games. $\mathscr{A}_2$ is given pp' and $t_1^*, \ldots, t_q^*, s_q$, where the t_j^* are PRG outputs if $b' = 0$ and random strings if $b' = 1$. $\mathscr{A}_2$ generates $M \twoheadleftarrow \mathsf{init}'(pp')$, $(pk, sk) \twoheadleftarrow \mathsf{KGen}$, and uses $t_1^*, \ldots, t_q^*, s_q$ to compute the forward-security challenge for $\mathscr{A}$, which corresponds to $\mathcal{G}_1$ if $b' = 0$, and $\mathcal{G}_2$ if $b' = 1$, implying (17), (18).

Game $\mathcal{G}_3$ is identical to $\mathcal{G}_2$ but, rather than setting $C_q \leftarrow \mathsf{Rand}(C_0; t'_1, \ldots, t'_q)$, we now set $C_q \leftarrow C_0$ where $C_0 \twoheadleftarrow \mathsf{Enc}(pk, M)$. By the re-randomization property of the encryption scheme, it must be the case that

$$\varDelta(C_0, \mathsf{Rand}(C_0; t'_1)) \leq \nu.$$

Since applying a function cannot increase statistical distance, this implies that

$$\varDelta(\mathsf{Rand}(C_0; t'_1, \cdots, t'_j), \mathsf{Rand}(C_0; t'_1, \cdots, t'_{j+1})) \leq \nu$$

for j=1, ..., q. Finally the triangle inequality implies that

$$\varDelta(C_0, \mathsf{Rand}(C_0, t'_1, \ldots, t'_q)) \leq q\nu,$$

justifying (19), (20).

Game $\mathcal{G}_4$ is identical to $\mathcal{G}_3$ except that we replace $C_0 \twoheadleftarrow \mathsf{Enc}(pk, M)$ with $C_0 \twoheadleftarrow \{0,1\}^l$. Since both games are perfectly simulatable by an adversary $\mathscr{A}_3$ in the IND\$-CPA game against $\mathcal{E}$ this implies (21), (22).

Game $\mathcal{G}_5$ is identical to $\mathcal{G}_4$ except that we replace the random strings $r_1, \ldots, r_q$ and C_0 with pseudorandom encryptions of M. Since both games are again perfectly simulatable by an adversary $\mathscr{A}_4$ in the IND\$-CPA game against $\mathcal{E}$ this implies (23), (24).

Game $\mathcal{G}_6$ is identical to $\mathcal{G}_5$ except that we replace the ciphertexts $C_j = \mathsf{Enc}(pk, M; t'_j)$ with $C_j = \mathsf{Rand}(C_0; t'_1, \ldots, t'_j)$ for $j = 1, \ldots, q$. The same statistical distance argument as above implies (25).

Game $\mathcal{G}_7$ is identical to $\mathcal{G}_6$ except that we replace the random strings t'_j with PRG outputs t_j. Again there exists an adversary $\mathscr{A}_5$ in the forward-security game against PRG' that can simulate both games, implying (26), (27).

Game $\mathcal{G}_8$ is identical to $\mathcal{G}_7$, except that we replace $C_0 \twoheadleftarrow \mathsf{Enc}(pk, M)$ with $C_0 \twoheadleftarrow \mathsf{Enc}(pk, s_0)$. As before, an adversary $\mathscr{A}_6$ in the IND-CPA game against $\mathcal{E}$ can perfectly simulate both games, implying (28), (29), and we conclude with the advantage term of (30).

```
Game $\mathcal{G}_0, \boxed{\mathcal{G}_1}$
$\underline{\mathsf{setup}}$
    $(pk, sk) \twoheadleftarrow \mathsf{KGen}$
    $(pp', \perp) \twoheadleftarrow \mathsf{setup}'$
    $pp \leftarrow (pk, pp')$
    $bk \leftarrow sk$
    $\mathbf{return}\ (pp, bk)$
$\underline{\mathsf{init}(pp)}$
    $s_0 \twoheadleftarrow \mathsf{init}'(pp')$
    $\boxed{M \twoheadleftarrow \mathsf{init}'(pp')}$
    $C_0 \twoheadleftarrow \mathsf{Enc}(pk, s_0)$
    $\boxed{C_0 \twoheadleftarrow \mathsf{Enc}(pk, M)}$
    $S_0 \leftarrow (s_0, C_0)$
    $\mathbf{return}\ (S_0)$
$\underline{\mathsf{challenge}(pp, S_0)}$
    $(s_0, C_0) \leftarrow S_0$
    $t_1, \ldots, t_q \leftarrow \mathsf{out}^q(\mathsf{next}'(pp', s_0))$
    $s_0, \ldots, s_q \leftarrow \mathsf{state}^q(\mathsf{next}'(pp', s_0))$
    $C_q \leftarrow \mathsf{Rand}(C_0; t_1, \ldots, t_q)$
    $S_q \leftarrow (s_q, C_q)$
    $r_1, \ldots, r_q \twoheadleftarrow (\{0,1\}^l)^q$
    $b' \leftarrow \mathscr{A}(pp, r_1, \ldots, r_q, S_q)$
    $\mathbf{return}\ (b' = 1)$
```

```
Game $\mathcal{G}_2, \boxed{\mathcal{G}_3}$
$\underline{\mathsf{setup}}$
    $(pk, sk) \twoheadleftarrow \mathsf{KGen}$
    $(pp', \perp) \twoheadleftarrow \mathsf{setup}'$
    $pp \leftarrow (pk, pp')$
    $bk \leftarrow sk$
    $\mathbf{return}\ (pp, bk)$
$\underline{\mathsf{init}(pp)}$
    $s_0 \twoheadleftarrow \mathsf{init}'(pp')$
    $M \twoheadleftarrow \mathsf{init}'(pp')$
    $C_0 \twoheadleftarrow \mathsf{Enc}(pk, M)$
    $S_0 \leftarrow (s_0, C_0)$
    $\mathbf{return}\ (S_0)$
$\underline{\mathsf{challenge}(pp, s_0)}$
    $(s_0, C_0) \leftarrow S_0$
    $t'_1, \ldots, t'_q \twoheadleftarrow (\{0,1\}^{l'})^q$
    $s_0, \ldots, s_q \leftarrow \mathsf{state}^q(\mathsf{next}'(pp', s_0))$
    $C_q \leftarrow \mathsf{Rand}(C_0; t'_1, \ldots, t'_q)$
    $\boxed{C_q \leftarrow C_0}$
    $S_q \leftarrow (s_q, C_q)$
    $r_1, \ldots, r_q \twoheadleftarrow (\{0,1\}^l)^q$
    $b' \leftarrow \mathscr{A}(pp, r_1, \ldots, r_q, S_q)$
    $\mathbf{return}\ (b' = 1)$
```

Fig. 15: Games $\mathcal{G}_0$, $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$ for proof of Theorem 3.

```
Game $\mathcal{G}_4$

$\underline{\mathsf{setup}}$
$(pk, sk) \twoheadleftarrow \mathsf{KGen}$
$(pp', \bot) \twoheadleftarrow \mathsf{setup}'$
$pp \leftarrow (pk, pp')$
$bk \leftarrow sk$
$\mathbf{return}\ (pp, bk)$
$\underline{\mathsf{init}(pp)}$
$s_0 \twoheadleftarrow \mathsf{init}'(pp')$
$M \twoheadleftarrow \mathsf{init}'(pp')$
$C_0 \twoheadleftarrow \{0,1\}^l$
$S_0 \leftarrow (s_0, C_0)$
$\mathbf{return}\ (S_0)$
$\underline{\mathsf{challenge}(pp, S_0)}$
$(s_0, C_0) \leftarrow S_0$
$t'_1, \ldots, t'_q \twoheadleftarrow (\{0,1\}^{l'})^q$
$s_0, \ldots, s_q \leftarrow \mathsf{state}^q(\mathsf{next}'(pp', s_0))$
$C_q \leftarrow C_0$
$S_q \leftarrow (s_q, C_q)$
$r_1, \ldots, r_q \twoheadleftarrow (\{0,1\}^l)^q$
$b' \leftarrow \mathscr{A}(pp, r_1, \ldots, r_q, S_q)$
$\mathbf{return}\ (b' = 1)$
```

```
Game $\mathcal{G}_5$, $\boxed{\mathcal{G}_6}$

$\underline{\mathsf{setup}}$
$(pk, sk) \twoheadleftarrow \mathsf{KGen}$
$(pp', \bot) \twoheadleftarrow \mathsf{setup}'$
$pp \leftarrow (pk, pp')$
$bk \leftarrow sk$
$\mathbf{return}\ (pp, bk)$
$\underline{\mathsf{init}(pp)}$
$s_0 \twoheadleftarrow \mathsf{init}'(pp')$
$M \twoheadleftarrow \mathsf{init}'(pp')$
$C_0 \twoheadleftarrow \mathsf{Enc}(pk, M)$
$S_0 \leftarrow (s_0, C_0, M)$
$\boxed{S_0 \leftarrow (s_0, C_0)}$
$\mathbf{return}\ (S_0)$
$\underline{\mathsf{challenge}(pp, S)}$
$(s_0, C_0, M) \leftarrow S_0$
$\boxed{(s_0, C_0) \leftarrow S_0}$
$t'_1, \ldots, t'_q \twoheadleftarrow (\{0,1\}^{l'})^q$
$s_0, \ldots, s_q \leftarrow \mathsf{state}^q(\mathsf{next}'(pp', s_0))$
$\mathbf{for}\ j = 1, \ldots, q$
  $C_j \leftarrow \mathsf{Enc}(pk, M; t'_j)$
  $\boxed{C_j \leftarrow \mathsf{Rand}(C_0; t'_1, \ldots, t'_j)}$
  $r_j \leftarrow C_{j-1}$
$S_q \leftarrow (s_q, C_q)$
$b' \leftarrow \mathscr{A}(pp, r_1, \ldots, r_q, S_q)$
$\mathbf{return}\ (b' = 1)$
```

Fig. 16: Games $\mathcal{G}_4$, $\mathcal{G}_5$ and $\mathcal{G}_6$ for proof of Theorem 3.

```
Game $\mathcal{G}_7$, $\boxed{\mathcal{G}_8}$
setup
$(pk, sk) \twoheadleftarrow \mathsf{KGen}$
$(pp', \perp) \twoheadleftarrow \mathsf{setup}'$
$pp \leftarrow (pk, pp')$
$bk \leftarrow sk$
return $(pp, bk)$
init$(pp)$
$s_0 \twoheadleftarrow \mathsf{init}'(pp')$
$M \twoheadleftarrow \mathsf{init}'(pp')$
$C_0 \twoheadleftarrow \mathsf{Enc}(pk, M)$
$\boxed{C_0 \twoheadleftarrow \mathsf{Enc}(pk, s_0)}$
$S \leftarrow (s_0, C_0)$
return $(S_0)$
challenge$(pp, s_0)$
$(s_0, C_0) \leftarrow S_0$
$t_1, \ldots, t_q \leftarrow \mathsf{out}^q(\mathsf{next}'(pp', s_0))$
$s_0, \ldots, s_q \leftarrow \mathsf{state}^q(\mathsf{next}'(pp', s_0))$
for $j = 1, \ldots, q$
  $C_j \leftarrow \mathsf{Rand}(C_0; t_1, \ldots, t_j)$
  $r_j \leftarrow C_{j-1}$
$S_q \leftarrow (s_q, C_q)$
$b' \leftarrow \mathscr{A}(pp, r_1, \ldots, r_q, S_q)$
return $(b' = 1)$
```

Fig. 17: Games $\mathcal{G}_7$ and $\mathcal{G}_8$ for proof of Theorem 3.

C Proofs for Section 4

C.1 Proof of Theorem 4

Proof. We firstly prove that $\overline{\mathsf{PRNG}} = (\overline{\mathsf{setup}}, \overline{\mathsf{init}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}})$ is ROB-secure against a non-backdoored attacker $\mathscr{A}$. To this end, we show that PRNG has both preserving and recovering security.

Lemma 3. *The PRNG* $\overline{\mathsf{PRNG}}$ *has* $(t, 2(5\epsilon_{ind} + 2\epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + 2k\epsilon_{rand}))$-PRE-*security.*

Proof. Let $\mathscr{A}$ be an adversary against $\overline{\mathsf{PRNG}}$ in $\mathrm{PRE}^{\mathscr{A}}_{\overline{\mathsf{PRNG}}}$. We argue by a series of game hops. The result follows from the following sequence of inequalities, with the corresponding security games $\mathcal{G}_1^{\mathscr{A}}, \ldots, \mathcal{G}_9^{\mathscr{A}}$ being shown in Figure 18, 19, and 20 (note that since we only modify $\overline{\mathsf{init}}$ and $\overline{\mathsf{next}}$ in the game hops, only the relevant code for these algorithms is shown in the figures).

$$\begin{aligned}
\frac{1}{2} + \frac{1}{2}\mathsf{Adv}^{\mathrm{pre}}_{\mathsf{PRNG}}(\mathscr{A}) &= \Pr\left[\mathcal{G}_0^{\mathscr{A}} \Rightarrow 1\right] && (31)\\
&\le \Pr\left[\mathcal{G}_1^{\mathscr{A}} \Rightarrow 1\right] + 2\epsilon_{ind} && (32)\\
&\le \Pr\left[\mathcal{G}_2^{\mathscr{A}} \Rightarrow 1\right] + 2\epsilon_{ind} + \epsilon_{pre} && (33)\\
&\le \Pr\left[\mathcal{G}_3^{\mathscr{A}} \Rightarrow 1\right] + 4\epsilon_{ind} + \epsilon_{pre} && (34)\\
&\le \Pr\left[\mathcal{G}_4^{\mathscr{A}} \Rightarrow 1\right] + 4\epsilon_{ind} + 2\epsilon_{pre} && (35)\\
&\le \Pr\left[\mathcal{G}_5^{\mathscr{A}} \Rightarrow 1\right] + 4\epsilon_{ind} + 2\epsilon_{pre} + \epsilon_{prg} && (36)\\
&\le \Pr\left[\mathcal{G}_6^{\mathscr{A}} \Rightarrow 1\right] + 4\epsilon_{ind} + 2\epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} && (37)\\
&\le \Pr\left[\mathcal{G}_7^{\mathscr{A}} \Rightarrow 1\right] + 4\epsilon_{ind} + 2\epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + k\epsilon_{rand} && (38)\\
&\le \Pr\left[\mathcal{G}_8^{\mathscr{A}} \Rightarrow 1\right] + 4\epsilon_{ind} + 2\epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + 2k\epsilon_{rerand} && (39)\\
&\le \Pr\left[\mathcal{G}_9^{\mathscr{A}} \Rightarrow 1\right] + 5\epsilon_{ind} + 2\epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + 2k\epsilon_{rerand} && (40)\\
&= \frac{1}{2} + 5\epsilon_{ind} + 2\epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + 2k\epsilon_{rerand}. && (41)
\end{aligned}$$

We begin by observing that $\mathcal{G}_0$ is identical to $\mathrm{PRE}^{\mathscr{A}}_{\mathsf{PRNG}}$, justifying (31). We then define $\mathcal{G}_1$, which is identical to $\mathcal{G}_0$, except we modify $\overline{\mathsf{init}}$ to replace $C_1 \twoheadleftarrow \mathsf{Enc}(pk, s)$ where $s \twoheadleftarrow \mathsf{init}(pp)$ with $C_1 \twoheadleftarrow \mathsf{Enc}(pk, 0^n)$. The following claim justifies (32).

Claim. $|\Pr\left[\mathcal{G}_1^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_0^{\mathscr{A}} \Rightarrow 1\right]| \le 2\epsilon_{ind}$

Proof. An adversary $\mathscr{A}_1$ in the IND-CPA game against $\mathcal{E}$ can perfectly simulate both games, by choosing $s \twoheadleftarrow \mathsf{init}(pp)$, submitting s and 0^n to his LR-oracle to receive ciphertext C_1, and setting $\overline{S} = (s, C_1, \mathsf{Enc}(pk, 0^n), \ldots, \mathsf{Enc}(pk, 0^n), \phi)$ for $\phi = 0$. Hence, the lemma follows.

Next we define $\mathcal{G}_2$, which is identical to $\mathcal{G}_1$ except we replace $(s', r) \leftarrow \mathsf{next}(pp, s)$ on line 7 of $\overline{\mathsf{next}}$ with $r \twoheadleftarrow \{0,1\}^l, s' \twoheadleftarrow \mathsf{init}(pp)$. The following claim justifies (33).

Claim. $|\Pr\left[\mathcal{G}_2^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_1^{\mathscr{A}} \Rightarrow 1\right]| \le \epsilon_{pre}$

Proof. An adversary $\mathscr{A}_2$ in $\mathrm{PRE}^{\mathscr{A}}_{\mathsf{PRNG}}$ against the underlying PRNG PRNG with challenge bit $\tilde{b}$ can perfectly simulate both games. $\mathscr{A}_2$ generates parameters, setting $\overline{pp} = (pp, pk)$ where pp is his challenge parameter. $\mathscr{A}_2$ then sets $\overline{S} = (s, C_1, \ldots, C_k, \phi)$ where $C_i = \mathsf{Enc}(pk, 0^n)$ for $i = 1, \ldots, k$, $\phi = 0$, and s represents the initial (and unknown) $s^0 \twoheadleftarrow \mathsf{init}(pp)$ in $\mathscr{A}_2$'s PRE challenge. $\mathscr{A}$ outputs a sequence of inputs, which $\mathscr{A}_2$ returns to its challenger, receiving back (s', r)

which $\mathscr{A}_2$ inserts at line 7, and uses to simulate the remainder of the challenge. At the end of the game, $\mathscr{A}_2$ outputs whatever bit $\mathscr{A}$ does. If $\tilde{b} = 0$ and $\mathscr{A}_2$ receives $(s', r) \leftarrow \mathsf{next}(pp, s)$ then this is a perfect run of $\mathcal{G}_1$, and if $\tilde{b} = 1$ and $\mathscr{A}_2$ receives $r \twoheadleftarrow \{0,1\}^l, s \twoheadleftarrow \mathsf{init}(pp)$ then this is a perfect run of $\mathcal{G}_2$. Hence, the claim follows.

Next we define $\mathcal{G}_3$, which is identical to $\mathcal{G}_2$ except we replace $C_0 \leftarrow \mathsf{Enc}(pk, s; r)$ with $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$ on line 9 of $\overline{\mathsf{next}}$. By a similar argument to before, a reduction to the IND-CPA security of $\mathcal{E}$ justifies the following claim, and thereby (34).

Claim. $\left| \Pr\left[\mathcal{G}_3^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_2^{\mathscr{A}} \Rightarrow 1\right] \right| \leq 2\epsilon_{ind}$

Now we define $\mathcal{G}_4$, which is identical to $\mathcal{G}_3$ except we replace $(s', r) \leftarrow \mathsf{next}(pp, s)$ on line 11 of $\overline{\mathsf{next}}$, with $r \twoheadleftarrow \{0,1\}^l, s' \twoheadleftarrow \mathsf{init}(pp)$. The following claim justifies (35).

Claim. $\left| \Pr\left[\mathcal{G}_4^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_3^{\mathscr{A}} \Rightarrow 1\right] \right| \leq \epsilon_{pre}$

Proof. This is again simulatable by an adversary $\mathscr{A}_3$ in $\mathrm{PRE}^{\mathscr{A}}_{\mathsf{PRNG}}$ against the underlying PRNG PRNG with challenge bit $\tilde{b}$. $\mathscr{A}_3$ again generates parameters and initial state as described in the previous reduction to PRE security. $\mathscr{A}_3$ ignores the inputs that $\mathscr{A}$ outputs, chooses $r \twoheadleftarrow \{0,1\}^l$, sets $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$, followed by $C_k \leftarrow C_{k-1}; \ldots ; C_1 \leftarrow C_0$ (taking us up to line 9). For $\mathscr{A}_3$'s PRE challenge, we view $s \twoheadleftarrow \mathsf{init}(pp)$ (unknown to $\mathscr{A}_3$)) in line 7 as $s^0 \twoheadleftarrow \mathsf{init}(pp)$ in the PRE challenge against the underlying PRNG PRNG. $\mathscr{A}_3$ returns the empty string to his challenger, and receives back (s', r), which $\mathscr{A}_3$ inserts at line 10, before continuing to simulate the rest of the challenge. If $\tilde{b} = 0$ and $\mathscr{A}_3$ receives $(s', r) \leftarrow \mathsf{next}(pp, s)$ then this is a perfect run of $\mathcal{G}_4$, and if $\tilde{b} = 1$ and $\mathscr{A}_3$ receives $r \twoheadleftarrow \{0,1\}^l, s \twoheadleftarrow \mathsf{init}(pp)$ then this is a perfect run of $\mathcal{G}_4$. Hence, the claim follows.

$\mathcal{G}_5$ is identical to $\mathcal{G}_4$, except we replace the PRG output $(b, r_1, \ldots, r_{2k}) \leftarrow \mathsf{PRG}(r)$ in line 11 of $\overline{\mathsf{next}}$ with truly random strings $(b, r_1, \ldots, r_{2k}) \twoheadleftarrow \{0,1\}^{2ku+1}$. Viewing $r \twoheadleftarrow \{0,1\}^l$ on line 10 as the seed for the PRG PRG, a straightforward reduction to PRG security justifies the following claim, and thereby (36).

Claim. $\left| \Pr\left[\mathcal{G}_5^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_4^{\mathscr{A}} \Rightarrow 1\right] \right| \leq \epsilon_{prg}$

$\mathcal{G}_6$ is identical to $\mathcal{G}_5$ except we replace the PRG output $\overline{r} \leftarrow \mathsf{PRG}'(r_1)$ on line 13 of $\overline{\mathsf{next}}$ with $\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}$. An identical argument to that above justifies the following claim and thereby (37).

Claim. $\left| \Pr\left[\mathcal{G}_6^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_5^{\mathscr{A}} \Rightarrow 1\right] \right| \leq \epsilon'_{prg}$

$\mathcal{G}_7$ is identical to $\mathcal{G}_6$ except we replace $C_i \leftarrow \mathsf{Rand}(C_i, r_{k+i})$ in line 19 of $\overline{\mathsf{next}}$ (where by construction each C_i is a re-randomised encryption of 0^n, and so a valid ciphertext), with $C_1 \leftarrow \mathsf{Enc}(pk, s; r_{k+1})$, (where $s \twoheadleftarrow \mathsf{init}(pp)$ in line 12), and $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_{k+1})$ for $i = 2, \ldots, k$. Since $\mathcal{E}$ is (t, δ)-strongly re-randomisable, the following claim follows, which also justifies (38).

Claim. $|\Pr\left[\mathcal{G}_7^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_6^{\mathscr{A}} \Rightarrow 1\right]| \leq k\epsilon_{rand}$

We note that the structure of $\overline{S}$ is now identical to that of a fresh state output by $\overline{\mathsf{init}}(pp)$.

$\mathcal{G}_8$ is identical to $\mathcal{G}_7$, except we replace $\mathsf{Rand}(C_i, r_i)$ on line 16 of $\overline{\mathsf{next}}$, (where again each C_i is a randomised encryption of 0^n, and so a valid ciphertext) with $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_i)$. As before, the re-randomisation property of $\mathcal{E}$ justifies the following claim, and thereby (39).

Claim. $|\Pr\left[\mathcal{G}_8^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_7^{\mathscr{A}} \Rightarrow 1\right]| \leq k\epsilon_{rand}$

Finally, $\mathcal{G}_9$ is identical to $\mathcal{G}_8$, except we replace $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_i)$ on line 16 of $\overline{\mathsf{next}}$ with $C_i \twoheadleftarrow \{0,1\}^m$. A straightforward reduction to the IND\$-CPA security of $\mathcal{E}$ justifies the following claim and (40).

Claim. $|\Pr\left[\mathcal{G}_9^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_8^{\mathscr{A}} \Rightarrow 1\right]| \leq k\epsilon_{ind}$

Notice that in $\mathcal{G}_9$, $\mathscr{A}$ receives $\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}$ and $\overline{S} \twoheadleftarrow \mathsf{init}(pp)$, regardless of the challenge bit b, implying $\Pr\left[\mathcal{G}_{10}^{\mathscr{A}} \Rightarrow 1\right] = \frac{1}{2}$. This justifies (41), and concludes the proof of robustness. □

Lemma 4. *The PRNG* $\overline{\mathsf{PRNG}}$ *has* $(t, q_r, \gamma^*, 2(3\epsilon_{ind} + \epsilon_{rec} + \epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + 2k\epsilon_{rand}))$-REC *security.*

Proof. Let ($\mathscr{A}$, $\mathscr{D}$) be an adversary/sampler pair in Game $\mathrm{REC}_{\overline{\mathsf{PRNG}}}^{\mathscr{A},\mathscr{D}}$ against $\overline{\mathsf{PRNG}}$. We argue by a series of game hops. The result follows from the following sequence of inequalities, with the corresponding security games $\mathcal{G}_1^{\mathscr{A}}, \ldots, \mathcal{G}_8^{\mathscr{A}}$ being shown in Figure 21, 22, and 23 (note that since we only modify $\overline{\mathsf{next}}$ in the game hops, only the relevant code for this algorithm is shown in the figures).

$$\begin{aligned}
\frac{1}{2} + \frac{1}{2}\mathsf{Adv}_{\overline{\mathsf{PRNG}}}^{\mathrm{rec}}(\mathscr{A}) &= \Pr[\mathcal{G}_0 \Rightarrow 1] && (42)\\
&\leq \Pr[\mathcal{G}_1 \Rightarrow 1] + \epsilon_{rec} && (43)\\
&\leq \Pr[\mathcal{G}_2 \Rightarrow 1] + 2\epsilon_{ind} + \epsilon_{rec} && (44)\\
&\leq \Pr[\mathcal{G}_3 \Rightarrow 1] + 2\epsilon_{ind} + \epsilon_{rec} + \epsilon_{pre} && (45)\\
&\leq \Pr[\mathcal{G}_4 \Rightarrow 1] + 2\epsilon_{ind} + \epsilon_{rec} + \epsilon_{pre} + \epsilon_{prg} && (46)\\
&\leq \Pr[\mathcal{G}_5 \Rightarrow 1] + 2\epsilon_{ind} + \epsilon_{rec} + \epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} && (47)\\
&\leq \Pr[\mathcal{G}_6 \Rightarrow 1] + 2\epsilon_{ind} + \epsilon_{rec} + \epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + k\epsilon_{rand} && (48)\\
&\leq \Pr[\mathcal{G}_7 \Rightarrow 1] + 2\epsilon_{ind} + \epsilon_{rec} + \epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + 2k\epsilon_{rand} && (49)\\
&\leq \Pr[\mathcal{G}_8 \Rightarrow 1] + 3\epsilon_{ind} + \epsilon_{rec} + \epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + 2k\epsilon_{rand} && (50)\\
&= \frac{1}{2} + 3\epsilon_{ind} + \epsilon_{rec} + \epsilon_{pre} + \epsilon_{prg} + \epsilon'_{prg} + 2k\epsilon_{rand} && (51)
\end{aligned}$$

```
$\mathcal{G}_0$, $\boxed{\mathcal{G}_1}$
$\overline{\mathsf{init}}(\overline{pp})$
1 : parse $\overline{pp}$ as $(pp, pk)$
2 : $s \twoheadleftarrow \mathsf{init}(pp)$
3 : $C_1 \twoheadleftarrow \mathsf{Enc}(pk, s)$
    $\boxed{C_1 \twoheadleftarrow \mathsf{Enc}(pk, 0^n)}$
4 : for $i = 2$ to $k$
5 :     $C_i \twoheadleftarrow \mathsf{Enc}(pk, 0^n)$
6 : $\phi \leftarrow 0$
7 : return $(s, C_1 \dots C_k, \phi)$
```

```
$\mathcal{G}_1$, $\boxed{\mathcal{G}_2}$
$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
...
6 : if $\phi = 1$
7 :     $(s, r) \leftarrow \mathsf{next}(pp, s)$
        $\boxed{s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l}$
8 :     $C_0 \leftarrow \mathsf{Enc}(pk, s; r)$
9 :     $C_k \leftarrow C_{k-1}; \dots ; C_1 \leftarrow C_0$
10 : $(s, r) \leftarrow \mathsf{next}(pp, s)$
11 : $(b, r_1, \dots, r_{2k}) \leftarrow \mathsf{PRG}(r)$
12 : if $b = 0$
13 :     $\overline{r} \leftarrow \mathsf{PRG}'(r_1)$
14 : else
15 :     for $i = 1$ to $k$
16 :         $C_i \leftarrow \mathsf{Rand}(C_i, r_i)$
17 :     $\overline{r} \leftarrow (C_1 \dots C_k)$
18 : for $i = 1$ to $k$
19 :     $C_i \leftarrow \mathsf{Rand}(C_i, r_{k+i})$
...
```

Fig. 18: Changes to the $\overline{\mathsf{init}}$ and $\overline{\mathsf{next}}$ algorithms in games $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$ in the proof of Lemma 3

We begin by observing that $\mathcal{G}_0$ is identical to $\mathrm{REC}^{\mathscr{D},\mathscr{A},q_r}_{\mathsf{PRNG},\gamma^*}$, justifying (42). Next we define $\mathcal{G}_1$, which is identical to $\mathcal{G}_0$ except we replace $(s', r) \leftarrow \mathsf{next}(pp, s)$ on line 7 of $\overline{\mathsf{next}}$ with $r \twoheadleftarrow \{0,1\}^l, s' \twoheadleftarrow \mathsf{init}(pp)$. The following claim justifies (43).

Claim. $|\Pr\left[\mathcal{G}_0^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_1^{\mathscr{A}} \Rightarrow 1\right]| \leq \epsilon_{rec}$

Proof. We observe that an adversary $\mathscr{A}_1$ in game $\mathrm{REC}^{\mathscr{D},\mathscr{A},q_r}_{\mathsf{PRNG},\gamma^*}$ against the underlying PRNG PRNG with challenge bit $\tilde{b}$ can perfectly simulate both games. $\mathscr{A}_1$ generates parameters, setting $\overline{pp} = (pp, pk)$ where pp is his challenge parameter, and passes these to $\mathscr{A}$. $\mathscr{A}_1$ simulates $\mathscr{A}$'s SAM oracle by querying his own, and passing the inputs to $\mathscr{A}$. Eventually, $\mathscr{A}$ outputs an initial state $\overline{S} = (s, C_1, \dots, C_k, \phi)$ and index d. $\mathscr{A}_1$ passes s, d to his own challenger, receiving back unused inputs $\mathbf{I}[k + d + 1 : q_r]$ and (s', r), which $\mathscr{A}_1$ inserts at line 7, and uses to simulate the remainder of the challenge. At the end of the game, $\mathscr{A}_1$ outputs whatever bit $\mathscr{A}$ does. If $\tilde{b} = 0$ and $\mathscr{A}_1$ receives $(s', r) \leftarrow \mathsf{next}(pp, s)$ then

```
$\mathcal{G}_3$, $\boxed{\mathcal{G}_4}$
$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
...
6 :  if $\phi = 1$
7 :     $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
8 :     $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$
9 :     $C_k \leftarrow C_{k-1}; \dots; C_1 \leftarrow C_0$
10 : $(s, r) \leftarrow \mathsf{next}(pp, s)$
     $\boxed{s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l}$
11 : $(b, r_1, \dots, r_{2k}) \leftarrow \mathsf{PRG}(r)$
12 : if $b = 0$
13 :    $\overline{r} \leftarrow \mathsf{PRG}'(r_1)$
14 : else
15 :    for $i = 1$ to $k$
16 :       $C_i \leftarrow \mathsf{Rand}(C_i, r_i)$
17 :    $\overline{r} \leftarrow (C_1 \dots C_k)$
18 : for $i = 1$ to $k$
19 :    $C_i \leftarrow \mathsf{Rand}(C_i, r_{k+i})$
...
```

```
$\mathcal{G}_5$, $\boxed{\mathcal{G}_6}$
$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
...
6 :  if $\phi = 1$
7 :     $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
8 :     $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$
9 :     $C_k \leftarrow C_{k-1}; \dots; C_1 \leftarrow C_0$
10 : $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
11 : $(b, r_1, \dots, r_{2k}) \twoheadleftarrow \{0,1\}^{2ku+1}$
12 : if $b = 0$
13 :    $\overline{r} \leftarrow \mathsf{PRG}'(r_1)$
        $\boxed{\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}}$
14 : else
15 :    for $i = 1$ to $k$
16 :       $C_i \leftarrow \mathsf{Rand}(C_i, r_i)$
17 :    $\overline{r} \leftarrow (C_1 \dots C_k)$
18 : for $i = 1$ to $k$
19 :    $C_i \leftarrow \mathsf{Rand}(C_i, r_{k+i})$
...
```

Fig. 19: Changes to the $\overline{\mathsf{next}}$ algorithm in games $\mathcal{G}_3$, $\mathcal{G}_4$, $\mathcal{G}_4$, and $\mathcal{G}_6$ in the proof of Lemma 3

this is a perfect run of $\mathcal{G}_0$, and if $\tilde{b} = 1$ and $\mathscr{A}_1$ receives $r \twoheadleftarrow \{0,1\}^l, s' \twoheadleftarrow \mathsf{init}(pp)$ then this is a perfect run of $\mathcal{G}_1$. Hence, the claim follows.

Next we define $\mathcal{G}_2$, which is identical to $\mathcal{G}_1$ except we replace $C_0 \leftarrow \mathsf{Enc}(pk, s; r)$ with $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$ on line 8 of $\overline{\mathsf{next}}$. By a similar argument to before, a reduction to the IND-CPA security of $\mathcal{E}$ justifies the following claim, and thereby (44).

Claim. $|\Pr\left[\mathcal{G}_1^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_2^{\mathscr{A}} \Rightarrow 1\right]| \leq 2\epsilon_{ind}$

Now we define $\mathcal{G}_3$, which is identical to $\mathcal{G}_2$ except we replace $(s', r) \leftarrow \mathsf{next}(pp, s)$ on line 10 of $\overline{\mathsf{next}}$, with $r \twoheadleftarrow \{0,1\}^l, s' \twoheadleftarrow \mathsf{init}(pp)$. The following claim justifies (45).

Claim. $|\Pr\left[\mathcal{G}_2^{\mathscr{A}} \Rightarrow 1\right] - \Pr\left[\mathcal{G}_3^{\mathscr{A}} \Rightarrow 1\right]| \leq \epsilon_{pre}$

Proof. This is again simulatable by an adversary $\mathscr{A}_2$ in $\mathrm{PRE}_{\mathsf{PRNG}}^{\mathscr{A}}$ against the underlying PRNG PRNG. $\mathscr{A}_2$ generates parameters, setting $\overline{pp} = (pp, pk)$ where

```
$\mathcal{G}_7$, $\boxed{\mathcal{G}_8}$
$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
...
6 :  if $\phi = 1$
7 :      $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
8 :      $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$
9 :      $C_k \leftarrow C_{k-1}; \dots; C_1 \leftarrow C_0$
10 : $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
11 : $(b, r_1, \dots, r_{2k}) \twoheadleftarrow \{0,1\}^{2ku+1}$
12 : if $b = 0$
13 :     $\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}$
14 : else
15 :     for $i = 1$ to $k$
16 :         $C_i \leftarrow \mathsf{Rand}(C_i, r_i)$
             $\boxed{C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_i)}$
17 :     $\overline{r} \leftarrow (C_1 \dots C_k)$
18 : for $i = 2$ to $k$
19 :     $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_{k+i})$
20 : $C_1 \leftarrow \mathsf{Enc}(pk, s; r_{k+1})$
...
```

```
$\mathcal{G}_9$
$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
...
6 :  if $\phi = 1$
7 :      $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
8 :      $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$
9 :      $C_k \leftarrow C_{k-1}; \dots; C_1 \leftarrow C_0$
10 : $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
11 : $(b, r_1, \dots, r_{2k}) \twoheadleftarrow \{0,1\}^{2ku+1}$
12 : if $b = 0$
13 :     $\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}$
14 : else
15 :     for $i = 1$ to $k$
16 :         $C_i \twoheadleftarrow \{0,1\}^m$
17 :     $\overline{r} \leftarrow (C_1 \dots C_k)$
18 : for $i = 2$ to $k$
19 :     $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_{k+i})$
20 : $C_1 \leftarrow \mathsf{Enc}(pk, s; r_{k+1})$
...
```

Fig. 20: Changes to the $\overline{\mathsf{next}}$ algorithm in games $\mathcal{G}_7$, $\mathcal{G}_8$, and $\mathcal{G}_9$ in the proof of Lemma 3

pp is his challenge parameter, passes these to $\mathscr{A}$, and simulates the oracle SAM using the code of the sampler $\mathscr{D}$. Eventually, $\mathscr{A}$ outputs state $(s, C_1, \dots, C_k, \phi)$ and some value d. $\mathscr{A}_2$ discards s and d, and keeps the remaining values to simulate the rest of the challenge. Again, for $\mathscr{A}_2$'s PRE challenge, we view $s \twoheadleftarrow \mathsf{init}(pp)$ (unknown to $\mathscr{A}_2$) in line 7 as $s^0 \twoheadleftarrow \mathsf{init}(pp)$ in the PRE challenge against the underlying PRNG PRNG. As described in the proof of Lemma 4, $\mathscr{A}_2$ returns the empty string to its challenger, and receives back (s', r), which $\mathscr{A}_2$ inserts at line 10. $\mathscr{A}_2$ then simulates the remainder of the challenge, again using the code of the sampler $\mathscr{D}$ to produce the unused inputs which are given to $\mathscr{A}$ at the end.

$\mathcal{G}_4$ is identical to $\mathcal{G}_3$ except we replace the PRG output $(b, r_1, \dots, r_{2k}) \leftarrow \mathsf{PRG}(r)$ in line 11 of $\overline{\mathsf{next}}$ with truly random strings $(b, r_1, \dots, r_{2k}) \twoheadleftarrow \{0,1\}^{2ku+1}$. Viewing $r \twoheadleftarrow \{0,1\}^l$ on line 10 of $\overline{\mathsf{next}}$ as the seed for the PRG PRG, a straightforward reduction to PRG security justifies the following claim, and thereby (46).

Claim. $|\Pr[\mathcal{G}_3^{\mathscr{A}} \Rightarrow 1] - \Pr[\mathcal{G}_4^{\mathscr{A}} \Rightarrow 1]| \leq \epsilon_{prg}$

$\mathcal{G}_5$ is identical to $\mathcal{G}_4$ except we replace the PRG output $\overline{r} \leftarrow \mathsf{PRG}'(r_1)$ on line 11 of $\overline{\mathsf{next}}$ with $\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}$. An identical argument to that above justifies the following claim, and thereby (47).

Claim. $|\Pr[\mathcal{G}_4^{\mathscr{A}} \Rightarrow 1] - \Pr[\mathcal{G}_5^{\mathscr{A}} \Rightarrow 1]| \leq \epsilon'_{prg}$

$\mathcal{G}_6$ is identical to $\mathcal{G}_5$ except we replace $C_i \leftarrow \mathsf{Rand}(C_i, r_{k+i})$ in line 19, with $C_1 \leftarrow \mathsf{Enc}(pk, s; r_{k+i})$, (where $s \twoheadleftarrow \mathsf{init}(pp)$ in line 10), and $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_{k+1})$ for $i = 2, \ldots, k$. Note that by construction each C_i is a re-randomisation of a valid ciphertext, and so is itself a valid ciphertext. This is due to lines 3, 4, 5 of the original algorithm, in which the adversarially chosen ciphertexts are checked for validity, and replaced if invalid. Thus the fact that $\mathcal{E}$ is (t, ϵ_{rand})-strongly re-randomisable justifies the following claim, and thereby (48).

Claim. $|\Pr[\mathcal{G}_5^{\mathscr{A}} \Rightarrow 1] - \Pr[\mathcal{G}_6^{\mathscr{A}} \Rightarrow 1]| \leq k\epsilon_{rand}$

We note that the structure of $\overline{S}$ is now identical to that of a fresh state output by $\overline{\mathsf{init}}(pp)$.

$\mathcal{G}_7$ is identical to $\mathcal{G}_6$, except we replace $\mathsf{Rand}(C_i, r_i)$ on line 16 of $\overline{\mathsf{next}}$, (where, by the same justification as above, each C_i is a valid ciphertext), with $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_i)$. As before, the fact that $\mathcal{E}$ is (t, ϵ_{rand})-strongly re-randomisable justifies the following claim, and thereby (49).

Claim. $|\Pr[\mathcal{G}_6^{\mathscr{A}} \Rightarrow 1] - \Pr[\mathcal{G}_7^{\mathscr{A}} \Rightarrow 1]| \leq k\epsilon_{rand}$

Finally, $\mathcal{G}_8$ is identical to $\mathcal{G}_7$, except we replace $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_i)$ on line 16 with $C_i \twoheadleftarrow \{0,1\}^m$. A straightforward reduction to the IND\$-CPA security of $\mathcal{E}$ justifies the following claim and thereby (50).

Claim. $|\Pr[\mathcal{G}_7^{\mathscr{A}} \Rightarrow 1] - \Pr[\mathcal{G}_8^{\mathscr{A}} \Rightarrow 1]| \leq \epsilon_{ind}$

Notice that in $\mathcal{G}_8$, $\mathscr{A}$ receives $\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}$ and $\overline{S} \twoheadleftarrow \mathsf{init}(pp)$, regardless of the challenge bit b, implying $\Pr[\mathcal{G}_8 \Rightarrow 1] = \frac{1}{2}$. This justifies (51), and concludes the proof. □

The robustness of $\overline{\mathsf{PRNG}}$ follows from combining the above two lemmas with Theorem 1. It remains to prove the success probability of the $\mathscr{B}$ algorithm in recovering previous output values.

Lemma 5. *For all refresh patterns* $\boldsymbol{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$*, where* a_i, b_i, ρ *are polynomial in the security parameter, for all distribution samplers* $\mathscr{D}$*, for all* $1 \leq i, j \leq \sum_{\nu=1}^{\rho} a_\nu$*, where* $i \neq j$*, it holds that* $\mathsf{Adv}_{\textsf{\textit{type}}}^{\text{bprng}}(\overline{\mathsf{PRNG}}, \boldsymbol{rp}, \mathscr{D}, i, j) \geq \delta(\boldsymbol{rp}, i, j)$ *where*

$$\delta(\boldsymbol{rp}, i, j) = \begin{cases} (1/4 - 2\epsilon_{prg} - a(\epsilon_{pre} + \epsilon_{rec})) & \text{if } j \leq i \ \wedge \ i_{ref} - j_{ref} + 1 \leq k \\ 0 & \text{otherwise} \end{cases}$$

$\boldsymbol{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$, $a = \sum_{\nu=1}^{\rho} a_\nu$,

$$i_{ref} \leftarrow \max_\sigma \left[\sum_{\nu=1}^{\sigma} a_\nu < i\right] \quad \text{and} \quad j_{ref} \leftarrow \max_\sigma \left[\sum_{\nu=1}^{\sigma} a_\nu < j\right].$$

```
$\mathcal{G}_0$, $\boxed{\mathcal{G}_1}$
$\underline{\overline{\mathsf{next}}(\overline{pp}, \overline{S})}$
...
if $\phi = 1$
    $(s, r) \leftarrow \mathsf{next}(pp, s)$
    $\boxed{s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l}$
    $C_0 \leftarrow \mathsf{Enc}(pk, s; r)$
    $C_k \leftarrow C_{k-1}; \ldots ; C_1 \leftarrow C_0$
  $(s, r) \leftarrow \mathsf{next}(pp, s)$
  $(b, r_1, \ldots, r_{2k}) \leftarrow \mathsf{PRG}(r)$
  if $b = 0$
    $\overline{r} \leftarrow \mathsf{PRG}'(r_1)$
  else
    for $i = 1$ to $k$
      $C_i \leftarrow \mathsf{Rand}(C_i, r_i)$
    $\overline{r} \leftarrow (C_1 \ldots C_k)$
  for $i = 1$ to $k$
    $C_i \leftarrow \mathsf{Rand}(C_i, r_{k+i})$
...
```

```
$\mathcal{G}_2$, $\boxed{\mathcal{G}_3}$
$\underline{\overline{\mathsf{next}}(\overline{pp}, \overline{S})}$
...
if $\phi = 1$
    $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
    $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$
    $C_k \leftarrow C_{k-1}; \ldots ; C_1 \leftarrow C_0$
  $(s, r) \leftarrow \mathsf{next}(pp, s)$
  $\boxed{s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l}$
  $(b, r_1, \ldots, r_{2k}) \leftarrow \mathsf{PRG}(r)$
  if $b = 0$
    $\overline{r} \leftarrow \mathsf{PRG}'(r_1)$
  else
    for $i = 1$ to $k$
      $C_i \leftarrow \mathsf{Rand}(C_i, r_i)$
    $\overline{r} \leftarrow (C_1 \ldots C_k)$
  for $i = 1$ to $k$
    $C_i \leftarrow \mathsf{Rand}(C_i, r_{k+i})$
...
```

Fig. 21: Changes to the $\overline{\mathsf{next}}$ algorithm in games $\mathcal{G}_0$, $\mathcal{G}_1$, $\mathcal{G}_3$, and $\mathcal{G}_3$ in the proof of Lemma 4

```
$\mathcal{G}_4$, $\boxed{\mathcal{G}_5}$
$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
$\ldots$
if $\phi = 1$
    $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
    $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$
    $C_k \leftarrow C_{k-1}; \ldots; C_1 \leftarrow C_0$
$s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
$(b, r_1, \ldots, r_{2k}) \twoheadleftarrow \{0,1\}^{2ku+1}$
if $b = 0$
    $\overline{r} \leftarrow \mathsf{PRG}'(r_1)$
    $\boxed{\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}}$
else
    for $i = 1$ to $k$
        $C_i \leftarrow \mathsf{Rand}(C_i, r_i)$
    $\overline{r} \leftarrow (C_1 \ldots C_k)$
for $i = 1$ to $k$
    $C_i \leftarrow \mathsf{Rand}(C_i, r_{k+i})$
$\ldots$
```

```
$\mathcal{G}_6$, $\boxed{\mathcal{G}_7}$
$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
$\ldots$
if $\phi = 1$
    $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
    $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$
    $C_k \leftarrow C_{k-1}; \ldots; C_1 \leftarrow C_0$
$s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
$(b, r_1, \ldots, r_{2k}) \twoheadleftarrow \{0,1\}^{2ku+1}$
if $b = 0$
    $\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}$
else
    for $i = 1$ to $k$
        $C_i \leftarrow \mathsf{Rand}(C_i, r_i)$
        $\boxed{C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_i)}$
    $\overline{r} \leftarrow (C_1 \ldots C_k)$
for $i = 2$ to $k$
    $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_{k+i})$
$C_1 \leftarrow \mathsf{Enc}(pk, s; r_{k+1})$
$\ldots$
```

Fig. 22: Changes to the $\overline{\mathsf{next}}$ algorithm in games $\mathcal{G}_4$, $\mathcal{G}_5$, $\mathcal{G}_6$, and $\mathcal{G}_7$ in the proof of Lemma 4

```
$\mathcal{G}_8$

$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
...
6 :   if $\phi = 1$
7 :     $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
8 :     $C_0 \leftarrow \mathsf{Enc}(pk, 0^n; r)$
9 :     $C_k \leftarrow C_{k-1}; \ldots; C_1 \leftarrow C_0$
10 :  $s \twoheadleftarrow \mathsf{init}(pp); r \twoheadleftarrow \{0,1\}^l$
11 :  $(b, r_1, \ldots, r_{2k}) \twoheadleftarrow \{0,1\}^{2ku+1}$
12 :  if $b = 0$
13 :    $\overline{r} \twoheadleftarrow \{0,1\}^{k \times m}$
14 :  else
15 :    for $i = 1$ to $k$
16 :      $C_i \twoheadleftarrow \{0,1\}^m$
17 :    $\overline{r} \leftarrow (C_1 \ldots C_k)$
18 :  for $i = 2$ to $k$
19 :    $C_i \leftarrow \mathsf{Enc}(pk, 0^n; r_{k+i})$
20 :  $C_1 \leftarrow \mathsf{Enc}(pk, s; r_{k+1})$
...
```

Fig. 23: Changes to the $\overline{\mathsf{next}}$ algorithm in game $\mathcal{G}_8$ in the proof of Lemma 4

Proof. Firstly note that if $\mathscr{B}$ is run with input values $j > i$ or $i_{ref} - j_{ref} \geq k$, the algorithm simply outputs $\perp$, and will hence not recover the output value $\overline{r}_j$.

We let b_i and b_j denote the value of b computed in line 11 of the $\overline{\mathsf{next}}$ algorithm, for the i-th and j-th output value $\overline{r}_i$ and $\overline{r}_j$, respectively. Note that for $\mathscr{B}$ to successfully recover $\overline{r}_j$, we must have that $b_i = 1$ (i.e. $\overline{r}_i = (C_1, \ldots, C_k)$) and $b_j = 0$ (i.e. $\overline{r}_j = \mathsf{PRG}'(r_1)$). However, if this is the case, note that $\mathscr{B}$ will output the correct value assuming the algorithm did not abort as a result of the above tests.

Recall that PRG is assumed to be ϵ_{prg}-secure, and note that, by Theorem 1, PRNG is $(\gamma^*, q_n \cdot (\epsilon_{pre} + \epsilon_{rec}))$-robust, where q_n is the number of Ror queries. It is straightforward to see that a polynomial time adversary will have advantage at most $2\epsilon_{prg} + a \cdot (\epsilon_{rec} + \epsilon_{pre})$ in distinguishing the values b_i and b_j from truly random values, where a is the total number of calls to $\overline{\mathsf{next}}$. Since $b_i = 1$ and $b_j = 0$ happens with probability $1/4$ if b_i and b_j were uniformly chosen, it follows that $\Pr[b_i = 1 \ \wedge \ b_j = 0] \geq 1/4 - 2\epsilon_{prg} - a(\epsilon_{pre} + \epsilon_{rec})$.

Hence, the lemma follows. □

D Proof of Theorem 5

The proof proceeds via a sequence of 4 claims.

Claim 1:

For all indices $j \geq 1$, all initial state values s_0 and all sequences $\bar{\iota}_{f(j-1)} = \iota_1, \iota_2, \dots, \iota_{f(j-1)}$ that may be output by $\mathscr{D}$, it holds that:

$$\sum_z \mathrm{CP}(S_{f(j)} | \bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)}, S_0 = s_0, pp = z) \Pr(pp = z) \leq \epsilon$$

Proof. For any well-behaved distribution sampler $\mathscr{D}$ and any such sequence of run outcomes we can construct a source $\mathscr{D}'(\varepsilon) = \mathscr{D}(\sigma_j)$ and a corresponding robustness adversary $\mathscr{A}$. The adversary $\mathscr{A}$ first uses $\iota_1, \iota_2, \dots, \iota_{f(j-1)}$, s_0, and the public parameter pp to compute $s_{f(j-1)}$. It then makes a set query to set the initial state to $s_{f(j-1)}$, makes the sequence of queries $q_{f(j-1)+1}, \dots, q_{f(j)}$ and then repeatedly calls next until r bits are returned. It then repeats this experiment with its own copy of $\mathscr{D}'$ together with pp and an initial state equal to $s_{f(j-1)}$. It then checks whether the r bits returned by the oracle match the ones that it computed. If they match it returns 0 otherwise it returns 1. It is then easy to see that:

$$\begin{aligned} &\mathsf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG}}(\mathscr{A}, \mathscr{D}') \\ &= \sum_z \mathrm{CP}(R_{f(j)}, R_{f(j)+1}, ..| \bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)}, S_0 = s_0, pp = z) \Pr(pp = z) - \frac{1}{2^r} \\ &\qquad \geq \sum_z \mathrm{CP}(S_{f(j)} | \bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)}, S_0 = s_0, pp = z) \Pr(pp = z) - \frac{1}{2^r} \end{aligned}$$

as required.

Claim 2:

For all indices $j \geq 1$, all initial state values s_0 and all sequences $\bar{s}_{f(j-1)} = s_{f(j-1)} \dots, s_{f(1)}$ it holds that:

$$\sum_z \mathrm{CP}(S_{f(j)} | \bar{S}_{f(j-1)} = \bar{s}_{f(j-1)}, S_0 = s_0, pp = z) \Pr(pp = z) \leq \epsilon$$

Proof. For any pp value z, any index j, and any initial state value s_0, let $\chi[z, j, s_0]$ be the set of I value sequences $\bar{\iota}_{f(j-1)} = \iota_1, \iota_2, \dots, \iota_{f(j-1)}$ that yield the sequence of states $\bar{s}_{f(j-1)} = s_{f(j-1)} \ldots, s_{f(1)}$. We then have:

$$\sum_z \mathrm{CP}(S_{f(j)}|\bar{S}_{f(j-1)} = \bar{s}_{f(j-1)}, S_0 = s_0, pp = z) \Pr(pp = z) =$$

$$\sum_z \sum_{\bar{\iota} \in \chi[z,j,s_0]} \mathrm{CP}(S_{f(j)}|\bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)}, S_0 = s_0, pp = z) \times \left(\frac{\Pr(\bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)})}{\sum_{\bar{\iota} \in \chi[z,j,s_0]} \Pr(\bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)})} \right) \Pr(pp = z) \quad (52)$$

Now for all $\bar{\iota} \in \chi[z, j, s_0]$ let

$$Q_{j,s_0}(\bar{\iota}_{f(j-1)}, z) = \frac{\Pr(\bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)})}{\sum_{\bar{\iota} \in \chi[z,j,s_0]} \Pr(\bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)})}$$

and $Q_{j,s_0}(\bar{\iota}_{f(j-1)}, z) = 0$ otherwise. By Claim 1 we then have that:

$$\sum_z \mathrm{CP}(S_{f(j)}|\bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)}, S_0 = s_0, pp = z) \Pr(pp = z) \leq \epsilon \sum_z \Pr(pp = z)\,.$$

Multiplying by $Q_{j,s_0}(\bar{\iota}_{f(j-1)}, z)$ on both sides, summing over $\bar{\iota}_{f(j-1)}$, and switching the order of addition on both sides yields:

$$\sum_z \sum_{\bar{\iota}_{f(j-1)}} \mathrm{CP}(S_{f(j)}|\bar{I}_{f(j-1)} = \bar{\iota}_{f(j-1)}, S_0 = s_0, pp = z) Q_{j,s_0}(\bar{\iota}_{f(j-1)}, z) \Pr(pp = z) \leq \epsilon \sum_z \sum_{\bar{\iota}_{f(j-1)}} Q_{j,s_0}(\bar{\iota}_{f(j-1)}, z) \Pr(pp = z)\,.$$

Now note that the LHS is equal to the LHS of Claim 2 (by equation (52)) and the RHS reduces to ϵ since for all z the sum of $Q_{j,s_0}(\bar{\iota}_{f(j-1)}, z)$ over all $\bar{\iota}_{f(j-1)}$ is 1, thereby proving Claim 2.

Claim 3:

For all indices $j \geq 1$ it holds that:

$$\mathrm{CP}(\bar{S'}_{f(j)}|pp) \leq \epsilon\, \mathrm{CP}(\bar{S'}_{f(j-1)}|pp)$$

Proof. Let $\bar{S'}_{f(j-1)} = \bar{S}_{f(j-1)}, S_{f(0)}$ and $\bar{s}'_{f(j-1)}$ represent any sequence $s_{f(j-1)} \ldots,$ $s_{f(1)}, s_{f(0)}$. Then by Claim 2 for all indices $j \geq 1$ and all sequences $\bar{s}'_{f(j-1)}$ we have:

$$\sum_z \mathrm{CP}(S_{f(j)}|\bar{S'}_{f(j-1)} = \bar{s}'_{f(j-1)}, pp = z) \Pr(pp = z) \leq \epsilon \sum_z \Pr(pp = z)$$

$$\sum_z \sum_{s_{f(j)}} \Pr(S_{f(j)} = s_{f(j)}|\bar{S'}_{f(j-1)} = \bar{s}'_{f(j-1)}, pp = z)^2 \Pr(pp = z) \leq \epsilon \sum_z \Pr(pp = z)$$

Now multiply both sides of the inequality by $\Pr(\bar{S'}_{f(j-1)} = \bar{s}'_{f(j-1)}|pp = z)^2$ and sum over $\bar{s}'_{f(j-1)}$.

$$\sum_{z} \sum_{\bar{s}'_{f(j-1)}} \sum_{s_{f(j)}} \Pr(S_{f(j)} = s_{f(j)}|\bar{S'}_{f(j-1)} = \bar{s}'_{f(j-1)}, pp = z)^2 \\ \times \Pr(\bar{S'}_{f(j-1)} = \bar{s}'_{f(j-1)}|pp = z)^2 \Pr(pp = z) \\ \leq \epsilon \sum_{z} \sum_{\bar{s}'_{f(j-1)}} \Pr(\bar{S'}_{f(j-1)} = \bar{s}'_{f(j-1)}|pp = z)^2 \Pr(pp = z)$$

Joining terms on the LHS yields:

$$\sum_{z} \sum_{\bar{s}'_{f(j)}} \Pr(\bar{S'}_{f(j)} = \bar{s}'_{f(j)}|pp = z)^2 \Pr(pp = z) \\ \leq \epsilon \sum_{z} \sum_{\bar{s}'_{f(j-1)}} \Pr(\bar{S'}_{f(j-1)} = \bar{s}'_{f(j-1)}|pp = z)^2 \Pr(pp = z)$$

which yields the desired result.

Claim 4:

$$\mathrm{CP}(S_{f(0)}|pp) \leq \epsilon$$

Proof. Consider a robustness adversary $\mathscr{A}'$ that proceeds as follows. It runs the algorithm init with fresh coins, to get an initial state and it then uses that state together with the public parameter pp to run next recursively until it produces r bits of output. It then makes sufficiently many calls to its next oracle until it gets another r bits of output. If the r bits returned by the oracle match the ones that it computed, it returns 0 otherwise it returns 1. Then its advantage is given by:

$$\mathsf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG}}(\mathscr{A}', \mathscr{D}) = \mathrm{CP}(R_{f(0)}, R_{f(0)+1}, ..|pp) - \frac{1}{2^r} \\ \geq \mathrm{CP}(S_{f(0)}|pp) - \frac{1}{2^r}$$

as required.

We can now come to the main result. Combining Claims 3 and 4 yields:

$$\mathrm{CP}(\bar{S'}_{f(j)}|pp) \leq \epsilon^{j+1} \text{ or equivalently } \tilde{\mathrm{H}}_2\left(\bar{S'}_{f(j)}|pp\right) \geq (j+1)\log\left(\frac{1}{\epsilon}\right),$$

and using the fact that for any two jointly distributed random variables Y and X it holds that $\tilde{\mathrm{H}}_\infty(Y|X) \geq \frac{1}{2}\tilde{\mathrm{H}}_2(Y|X)$, we obtain:

$$\tilde{\mathrm{H}}_\infty\left(\bar{S'}_{f(j)}|pp\right) \geq \frac{j+1}{2}\log\left(\frac{1}{\epsilon}\right).$$

Now by applying the chain rule for min entropy on $\bar{S'}_{f(j)}$ and any output value $R_{f(j)+k}$ where $k \geq 0$ we get the desired result:

$$\begin{aligned}\tilde{\mathrm{H}}_\infty\left(\bar{S'}_{f(j)}|R_{f(j)+k}, pp\right) &\geq \tilde{\mathrm{H}}_\infty\left(\bar{S'}_{f(j)}|pp\right) - \mathrm{Supp}\left(R_{f(j)+k}\right) \\ &\geq \frac{j+1}{2}\log\left(\frac{1}{\epsilon}\right) - \min(n,l).\end{aligned}$$

□