On the Possibility of a Backdoor in the Micali-Schnorr Generator

Hannah Davis* Matthew Green† Nadia Heninger* Keegan Ryan* Adam Suhl*

Abstract

In this paper, we study both the implications and potential impact of backdoored parameters for two RSA-based pseudorandom number generators: the ISO-standardized Micali-Schnorr generator and a closely related design, the RSA PRG. We observe, contrary to common understanding, that the security of the Micali-Schnorr PRG is not tightly bound to the difficulty of inverting RSA. We show that the Micali-Schnorr construction remains secure even if one replaces RSA with a publicly evaluatable PRG, or a function modeled as an efficiently invertible random permutation. This implies that any cryptographic backdoor must somehow exploit the algebraic structure of RSA, rather than an attacker's ability to invert RSA or the presence of secret keys. We exhibit two such backdoors in related constructions: a family of exploitable parameters for the RSA PRG, and a second vulnerable construction for a finite-field variant of Micali-Schnorr. We also observe that the parameters allowed by the ISO standard are incompletely specified, and allow insecure choices of exponent. Several of our backdoor constructions make use of lattice techniques, in particular multivariate versions of Coppersmith's method for finding small solutions to polynomials modulo integers.

1 Introduction

In 2013, a collection of leaks due to Edward Snowden revealed the existence of a large-scale U.S. government effort called the SIGINT Enabling Project., intended to compromise the integrity of cryptographic systems. Equipped with a $200M annual budget, the project sought to "insert vulnerabilities into commercial encryption systems" and to "influence policies, standards and specification for commercial public key technologies" [SIG13]. These leaks also revealed that the U.S. National Security Agency authored and maintained sole editorial control of the 2005 ISO 18031 standard on random bit generation [PLS13], a standard that was largely incorporated into the U.S. ANSI X9.82 standard. The ANSI standard in turn forms the basis for the U.S. National Institute for Standards and Technology's NIST Special Publication 800-90A, which defines requirements for random bit generation in government-approved cryptographic products.

Even before the Snowden leaks, the NIST/ANSI and ISO standards have drawn scrutiny for their inclusion of a number-theoretic PRG known as the Dual Elliptic Curve Deterministic Random Bit Generator (Dual EC DRBG), a construction that is exploitable by a party that generates the system public parameters and retains a secret trapdoor [SF07]. The Snowden leaks inspired renewed investigation of this standard and its deployment, revealing that Dual EC was more widely deployed than many academic researchers had realized. Moreover, later investigation revealed that TLS and IPsec implementations incorporating Dual EC [CNE+14, CMG+16] also made specific

*University of California San Diego, hdavis@ucsd.edu, nadiah@cs.ucsd.edu, {kryan,asuhl}@ucsd.edu
†Johns Hopkins University, mgreen@cs.jhu.edu

implementation decisions that coincidentally rendered them practically exploitable by an adversary who possesses the Dual EC trapdoor.

Even if the standardized parameters were not *deliberately* backdoored, the mere possibility of such parameters poses a threat to users of a standardized PRG. In one noteworthy case, an undocumented implementation of Dual EC in Juniper NetScreen's firewalls appears to have been exploited in practice; in 2012 an outside group compromised the NetScreen codebase and replaced Juniper's Dual EC constants with parameters of their own devising [CMG+16]. These parameters were in place for over three years, presumably enabling the outside group to decrypt the communications of Juniper NetScreen customers, which at the time included the U.S. Federal Government. Demonstrating the existence, or ruling out the possibility, of methods to backdoor the parameter generation process is the only way to mitigate the risk of parameter substitution attacks which otherwise undermine the security of commercial encryption technology.

The MS DRBG generator. While Dual EC is the only number-theoretic generator adopted as a NIST standard, the current draft of ISO 18031 (and early drafts of the ANSI X9.82 standard) also include a second public-key generator that has received surprisingly little analysis. Based on a design by Micali and Schnorr [MS90, MS91], the MS DRBG algorithm is a pseudorandom number generator whose security is purportedly related to the hardness of breaking RSA. In brief, the algorithm is instantiated using a state s_0 and an RSA public key (N, e), and at each stage the algorithm applies the RSA function to the state to obtain an integer $z_{i+1} = s_i^e \bmod N$. The most significant bits of z_{i+1} become the new state s_{i+1} for the next iteration of the algorithm, and the least significant bits of z_{i+1} are the output b_{i+1} of the pseudorandom number generator for that iteration.

While RSA (and Rabin)-based pseudorandom generators have been studied in the academic literature for many years [BBS86, MS90, MS91, FS97, FS00, SPW06], two aspects of the MS DRBG standard draw attention. First, in contrast with common practice for RSA-based generators, the generator outputs as many as $1 - 2/e$ of the bits (or up to 864 bits for a 1024-bit modulus with a claimed 80-bit security level.)[1] More critically, the MS DRBG standard includes a design choice that is reminiscent of the Dual EC generator: namely, it incorporates a series of recommended public parameters that are intended to be used in production as the modulus N. As with Dual EC, the provenance of these moduli is not documented in the standard. However, correspondence from the ANSI standards process (revealed under the Freedom of Information Act [Nat15]) supports the conclusion that both Dual EC and MS DRBG were authored by the National Security Agency, which also generated the parameters for both specifications. Unlike the Dual EC parameters (which could conceivably have been generated such that the generating party would not learn a trapdoor) according to the standard the MS DRBG moduli are the product of primes p, q chosen by the standard author. The NSA's knowledge of this secret factorization calls into question the security of the generator when used in a setting where the NSA is adverse to its user.

MS DRBG has not, to our knowledge, been used in any real-world systems. However, several surprisingly widespread implementations of Dual EC DRBG (e.g., Juniper's NetScreen implementation) were unknown to researchers and were only discovered by chance long after the deprecation of Dual EC. MS DRBG, though still in the ISO standard, has received far less attention than Dual

[1] Previous RSA-based generators (including some early drafts of MS DRBG) recommend only to output $\lg \lg N$ bits at each RSA operation. MS DRBG's larger output is justified by a novel pseudorandomness assumption introduced by Micali and Schnorr [MS91].

EC from the research community. It is possible there were (or are) MS DRBG implementations used in production that have not received public scrutiny.

Given the known vulnerabilities in Dual EC DRBG and the (allegedly) identical provenance of MS DRBG, it is therefore reasonable to ask whether MS DRBG is vulnerable to an analogous attack. Concretely:

> *Does knowledge of the factors of (or malicious construction of) the recommended moduli imply a practical attack on the MS DRBG generator?*

This question is surprisingly difficult to answer. While the literature is replete with studies of RSA-based generators, the majority of this work naturally assumes that the factorization of N is kept secret. In that setting, standard results on RSA hardcore bits can be used to argue the indistinguishability of generator output. Clearly such arguments no longer apply in settings where the factorization is *known* to the attacker. And yet in contrast to many other RSA-based constructions, knowledge of the factorization does not point to an obvious attack strategy against MS DRBG or similar RSA-based generators. Such gaps between "best reduction" and "best attack" are hardly unknown in the literature. We argue, however, that the history and provenance of the MS DRBG standard make it worthy of a closer look.

Our results. In this paper, we study both the implications and potential impact of backdoored parameters for the Micali-Schnorr generator and for the RSA PRG, a closely related design that was never standardized. To our knowledge, we are the first to identify vulnerabilities in these algorithms from this perspective in the literature.

First, we observe that the security of the Micali-Schnorr PRG is not tightly bound to the difficulty of inverting RSA. We show that the Micali-Schnorr construction remains secure even if one replaces RSA with a publicly evaluatable PRG or an ideal (and efficiently invertible) random permutation. This implies that any cryptographic backdoor must somehow exploit the algebraic structure of RSA, rather than an attacker's ability to merely invert RSA or the presence of secret keys. We exhibit two such backdoors in related constructions: a family of exploitable parameters for the RSA PRG, and a second vulnerable construction for a finite-field variant of Micali-Schnorr. We also observe that the parameters allowed by the ISO standard are incompletely specified, and allow insecure choices of exponent. Several of our backdoor constructions make use of lattice techniques, in particular multivariate versions of Coppersmith's method for finding small solutions to polynomials modulo integers. We evaluate the impact of our attacks in the context of network protocols and find that the ISO weak exponent vulnerability would be exploitable in the context of IPsec.

Ultimately we were unsuccessful in fully solving the question we set out to answer, that of either finding an efficiently exploitable backdoor for the Micali-Schnorr generator or ruling out the possibility, but we hope that this work will bring more attention to this unsolved problem and point the way to potentially fruitful cryptanalytic advances.

1.1 Technical Overview

Our goal in this work is to evaluate the hypothesis that the standardization of ISO/ANSI MS DRBG may represent an intentional attempt to subvert cryptographic systems. This possibility is intriguing for two different reasons: first, a better understanding would offer new historical

context on the development of public-key standards and the intentions of nation-state cryptologic agencies. From a technical perspective, detailed investigation of this question might uncover the existence of heretofore non-public attacks on public-key cryptosystems. While factoring-based PRG constructions are some of the earliest work in this area, they appear to have received surprisingly little attention from a modern perspective, and this question shines new light on the tightness of the connection between the hardness assumptions and the security of the output.

Our approach is to assume the worst case: that the authors of the standard intentionally retained the factorization of each recommended modulus (or maliciously generated these moduli), and additionally possessed techniques that enabled them to practically exploit this knowledge. From this starting point we then attempt to "re-derive" the necessary techniques and to evaluate whether they can be used in practical settings. Our primary focus in this work is on techniques that enable state recovery given some quantity of generator output, since knowledge of the internal generator state would enable a passive attacker to obtain future generator output and possibly compromise the security of encryption protocols.

Indistinguishability, state recovery and backtracking resistance. Like most standardized generators, MS DRBG uses an iterated construction. The generator is initially seeded with a state s_0 that is repeatedly updated through application of a purported one-way function, while the same function is also used to produce output bits. As with any PRG, knowledge of previous outputs must not provide an attacker with a meaningful advantage in predicting future output bits. This can only be achieved if all internal states remain secret until the generator is reseeded, since knowledge of any state permits the prediction of all future states and outputs. Both ISO and ANSI require an even stronger security property: the compromise of any intermediate state s_i must not enable prediction or distinguishing of generator outputs from *previous* cycles. We show that neither MS DRBG nor RSA PRG achieve this property when the factorization of N is known.

Eliminating the obvious. A natural approach for a subversion attacker [2] to break the security of the PRGs is to simply invert the RSA function to recover some internal generator state after observing generator output. However, in MS DRBG (as well as more traditional RSA and Rabin-based generators) this direct approach fails because the construction does not output all bits of RSA function output.

While the security assumptions that underlie the security proof for MS DRBG [MS91] are clearly false if the factorization of the modulus N is known, straightforward attempts to reverse the security reduction are also futile, for a similar reason. Micali and Schnorr's reduction relies on two elements: a novel indistinguishability assumption for partial RSA outputs (repeated herein as Assumption 1), which is combined with rejection sampling and the algorithm of Alexi, Chor, Goldreich, and Schnorr [ACGS84] (in Theorem 1) to reduce this to the hardness of the RSA problem. An adversary who can falsify Assumption 1 still does not obtain all bits of the ciphertext from the generator's output. Even when the attacker has access to an inversion oracle, both sides of the reduction are efficient to solve, so even reversing this portion of the reduction is unlikely to lead to a practical subversion attack. Moreover, the running time of this reduction algorithm for parameters of practical interest is much more expensive than simply brute forcing the unknown state would be, which makes Theorem 1 vacuous in the case of MS DRBG.[3]

[2] An attacker with backdoor information; see Section 5.1.

[3] Naturally ISO has specified parameters so that brute force and collision attacks against the state are infeasible.

Eliminating black box attacks. Given the above, we turn our attention to whether being able to falsify Assumption 1 in a black box way suffices for a distinguishing attack on MS DRBG. Stated broadly, this assumption implies that for some length-expanding function $F : \{0,1\}^k \to \{0,1\}^n$ with $k \ll n$, the ensemble $\{b_1, b_2, \ldots, b_h\}$ is indistinguishable from random, when each $b_i \leftarrow F(s_{i-1}) \mod 2^{n-k}$ and $s_i \leftarrow \lfloor F(s_{i-1})/2^{n-k} \rfloor$ (with s_0 a random k-bit string). It is relatively easy to argue this assumption holds when F is itself a PRG (or is modeled as a random function or permutation on $\{0,1\}^n$): indeed, we prove this in Section 4. We observe that the MS PRG construction instantiated with such a random permutation remains secure even when the inverse oracle is available to an adversary, or when the function contains no secrets or keys. Indeed, this construction is analogous to several common PRG constructions based on hash functions. The theorems we prove are straightforward, and their implications are obvious in retrospect, but they encode observations about the security of the MS DRBG construction that do not appear to have been formalized in the literature.

The main question then is whether such an assumption can hold when F is realized via the RSA function in the unusual setting where the factorization is known. These results suggest that any attacks on the MS or RSA PRGs must exploit *algebraic* properties of RSA and modular exponentiation, rather than being able to take advantage of factorization in a black-box way.

Algebraic attacks. After eliminating the possibility of an "obvious" factoring-based backdoor, we give several constructions of candidate backdoor constructions and algorithmic weaknesses for RSA PRG and MS PRG that exploit algebraic properties of modular exponentiation. Most of our state recovery attacks make use of lattice-based techniques, in particular variants of multivariate Coppersmith's method. A straightforward application of Coppersmith-type methods to break MS and RSA PRG is ruled out by the ISO parameters; such attacks seem unlikely to allow recovery of a state larger than n/e bits for generic parameters (for RSA exponent e), while the ISO parameters set the state size at $2n/e$ bits.

We give backdoor constructions that introduce additional structure that results in feasible attacks that work beyond these known bounds. For RSA PRG, we show how to efficiently generate RSA moduli that embed cyclic and linear recurrence relations in the PRG output and how to hide these recurrences in the factorization of N; we also give an algorithm exploiting these recurrences for an efficient full state recovery attack from parameter ranges that were not known to be previously exploitable. Unfortunately, extending this idea to MS PRG does not seem to result in an efficiently exploitable backdoor without some further structure that allows us to simplify the exponentially many polynomial terms generated by the recurrence relations. We illustrate such an algebraic structure by showing that a variant of MS PRG defined over small-characteristic finite fields is trivially broken.

Finally, we show that the existence of RSA decryption exponents allows the efficient generation of apparently unnoticed weak exponent choices for MS PRG that are not ruled out by the parameters in the ISO standard. These weak exponents allow an efficient state recovery attack for the output lengths and state sizes in the ISO standard from a single PRG output block. We further observe that if this PRG were used to generate nonces and secret keys in the IPsec protocol (as Dual EC was in real-world implementations), this attack would allow an efficient state recovery attack from a single handshake nonce generated from raw PRG output.

Future directions. The problem of designing an efficient backdoor for the Micali-Schnorr scheme has floated around the cryptographic community as an open problem since at least 2013 [Gre13], with little success.

In this work, we rule out some obvious-seeming approaches that are dead ends, and illuminate some potentially fruitful directions for future exploration. In the end, we leave the question of identifying or ruling out an efficiently exploitable general backdoor in the Micali Schnorr algorithm unsolved in this paper. Ultimately, a solution to this problem may involve new ideas in the cryptanalysis of RSA.

2 Background

The ISO, NIST, and ANSI standards refer to the (pseudo)random number generation algorithms they describe as *random bit generators* (RBGs) and to deterministic pseudorandom number generation algorithms as *deterministic random bit generators* (DRBGs).

ISO-18031 lists a number of (informal) security requirements for RBGs. We list the most relevant of these below, and mark the exact text from the standard in quotes. We have given names to these properties for future convenience.

Indistinguishability "Under reasonable assumptions, it shall not be feasible to distinguish the output of the RBG from true random bits that are uniformly distributed." Indistinguishability has been extensively studied in the academic literature [Yao82, BM82].

State compromise resistance "The RBG shall not leak relevant secret information (e.g., internal state of a DRBG) through the output of the RBG." While indistinguishability is the main requirement for a PRG in the academic literature, the focus of many practical attacks is a full state compromise [CNE+14, CMG+16, CGH18].

The following properties[4] are listed as optional requirements:

Backtracking resistance "Given all accessible information about the RBG (comprising some subset of inputs, algorithms, and outputs), it shall be computationally infeasible (up to the specified security strength) to compute or predict any previous output bit."

Prediction resistance "Given all accessible information about the RBG (comprising some subset of inputs, algorithms, and outputs), it shall be infeasible (up to the specified security strength) to compute or predict any future output bit at the time that [prediction resistance] was requested."

We will use the term "PRG" to refer to the algorithms in this paper, and the term DRBG when we specifically reference the ISO standard. There have also been extensive academic efforts to formalize requirements for pseudorandom number generation under various attack models including recovery from state compromise [DPR+13].

[4]The standard calls these properties "backward secrecy" and "forward secrecy", but in the opposite way from the academic literature; we rename these properties to avoid confusion.

2.1 The RSA PRG

The ISO standard introduces MS DRBG as a variant of the "so-called RSA generator", which iteratively applies RSA encryption to a starting seed to generate a sequence of states and outputs some least significant bits of each state as output. Micali and Schnorr [MS90] name this algorithm "the 'incestuous' generator".

We summarize this algorithm in Algorithm 1 below. Let N be an integer RSA modulus of length $\lg N = n$. Let k be the length of the PRG output on a single iteration, so we have $k < n$. The length of the state is $r = n$ bits. e will be a positive integer that is the RSA exponent; in order for (N, e) to be valid RSA parameters e should be relatively prime to $\varphi(N)$. (The choice $e = 2$ is the Rabin generator used in Blum-Blum-Shub [BBS86].)

Algorithm 1: RSA PRG

Input : A number of iterations h
Output: hk pseudorandom bits

Sample initial state $s_0 \leftarrow \$[1, N]$ using truly random coins.
for $i \leftarrow 1$ **to** h **do**
  $s_i \leftarrow s_{i-1}^e \bmod N$
  $b_i \leftarrow s_i \bmod 2^k$
end
Output the concatenation $b_1||b_2||\ldots||b_h$.

There is a simple formula for the ith state in terms of the initial state:

$$s_i \equiv s_0^{e^i} \bmod N$$

Output Length There are several choices of output lengths discussed in the literature on this PRG.

- $k = \lg n$. Micali and Schnorr [MS91] show that the RSA PRG is secure for $k = \lg n$ bits of the RSA function, based on the Alexi-Chor-Goldreich-Schnorr theorem [ACGS84] that these bits are hardcore.
- $k = 1$. Fischlin and Schnorr [FS00] improve the running time of the ACGS reduction and use this running time to propose concrete parameters of $k = 1$ bit of output with a 1000-bit modulus N.
- $k = (1/2 - 1/e - \epsilon - o(1))n$. Steinfeld, Pieperzyk, and Wang [SPW06] prove the security of outputting more bits under the hardness of improving on the Coppersmith bound for solving polynomials modulo RSA moduli [Cop97].

2.2 The Micali-Schnorr PRG (MS PRG)

Like the RSA PRG, the Micali-Schnorr PRG iteratively applies the RSA function, but it separates the bits used to generate the next state from the bits that are output. It splits each RSA output into its most and least significant bits. The least significant bits become PRG output, while the most significant bits become the RSA input for the PRG's next iteration.

Let N be an integer RSA modulus of length $\lg N = n$. Let k be the length of the PRG output on a single iteration, so we have $k < n$. r will be the internal state length; for the Micali-Schnorr algorithm we have $r = n - k$. e will be a positive integer that is the RSA exponent; this is specified so that e is relatively prime to $\varphi(N)$.

2.2.1 Micali-Schnorr as published in their papers.

There are two published versions of the Micali-Schnorr paper. The first version, "Efficient, Perfect Random Number Generators," appeared in CRYPTO '88 [MS90]. The journal version of the paper, "Efficient, Perfect Polynomial Random Number Generators," was published in the Journal of Cryptology in 1991 [MS91].

The algorithm for the Micali-Schnorr PRG as published in the original paper takes an input h and iterates the algorithm h times to output hk pseudorandom bits. Micali and Schnorr refer to this construction as the "sequential polynomial generator of the weaning type."

Algorithm 2: The Micali-Schnorr algorithm

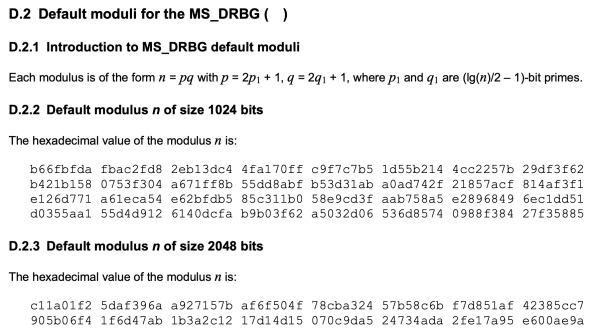
Input : A number of iterations h
Output: hk pseudorandom bits

Sample initial state $s_0 \leftarrow \$[1, N2^{-k}]$ using truly random coins.
for $i \leftarrow 1$ **to** h **do**
  $z_i \leftarrow s_{i-1}^e \mod N$
  $b_i \leftarrow z_i \mod 2^k$
  if *version from [MS90]* **then**
    $s_i \leftarrow \lfloor z_i 2^{-k} \rfloor + 1$;
  else if *ISO-18031 version* **then**
    $s_i \leftarrow \lfloor z_i 2^{-k} \rfloor$;
end
Output $b_1 || b_2 || \ldots || b_h$.

Output length. In the original paper, Micali and Schnorr discuss the following choices for k and n.

- $k = O(\lg n) = O(\lg \lg N)$ They list this as a suitable choice for the PRG. The algorithm in Micali and Schnorr's reduction runs in time polynomial in $2^k n \epsilon^{-1}$ so if the PRG is insecure for $k = O(\lg n)$ then this gives a polynomial time RSA decryption algorithm.
- $k = O(n^{1/3})$ They argue that this choice is suitable by comparing the resulting reduction time to the running time of the number field sieve for factoring, and note that if the algorithm is insecure for this parameter, it would beat the number field sieve.
- $k = n(1 - 1/e)$ is clearly insecure. For this choice, the security proof does not apply because $s_0^e < N$, so no mod operation occurs and "RSA" decryption is easy in this case.
- $k = n(1 - 2/e)$ is left as an open question, but Micali and Schnorr promote this choice as one that would produce an efficient generator if indeed it is secure. The security of this choice

is based on the indistinguishability of RSA encryptions of short ($n - k$-bit) plaintexts from random integers modulo N. This is the value chosen by the ISO standard.



D.2 Default moduli for the MS_DRBG ( )

D.2.1 Introduction to MS_DRBG default moduli

Each modulus is of the form $n = pq$ with $p = 2p_1 + 1$, $q = 2q_1 + 1$, where p_1 and q_1 are (lg(n)/2 – 1)-bit primes.

D.2.2 Default modulus *n* of size 1024 bits

The hexadecimal value of the modulus n is:

```
b66fbfda fbac2fd8 2eb13dc4 4fa170ff c9f7c7b5 1d55b214 4cc2257b 29df3f62
b421b158 0753f304 a671ff8b 55dd8abf b53d31ab a0ad742f 21857acf 814af3f1
e126d771 a61eca54 e62bfdb5 85c311b0 58e9cd3f aab758a5 e2896849 6ec1dd51
d0355aa1 55d4d912 6140dcfa b9b03f62 a5032d06 536d8574 0988f384 27f35885
```

D.2.3 Default modulus *n* of size 2048 bits

The hexadecimal value of the modulus n is:

```
c11a01f2 5daf396a a927157b af6f504f 78cba324 57b58c6b f7d851af 42385cc7
905b06f4 1f6d47ab 1b3a2c12 17d14d15 070c9da5 24734ada 2fe17a95 e600ae9a
```

Figure 1: A portion of ISO 18031 Appendix D.2 showing the default 1024-bit modulus and a portion of the 2048-bit modulus [Int11].

2.2.2 ISO/IEC 18031 Micali-Schnorr

The Micali-Schnorr algorithm was standardized in ISO/IEC 18031 as a deterministic random bit generator, named MS DRBG, alongside the Dual EC DRBG design. Dual EC was removed from ISO 18031 in a 2014 Technical Corrigendum.

The version of the Micali-Schnorr algorithm that appears in ISO/IEC 18031 [Int11] differs in one minor respect from the academic publication. Specifically, it alters line 6 so that $s_i \leftarrow \lfloor z_i 2^{-k} \rfloor$. (That is, it does not increment the result.) It isn't clear what effect, if any, this change has on the security of the scheme, and it is not documented in either publication. In personal communication, Micali told us he does not recall the reason for the original choice or the change.

Output length. The ISO standard requires that the output length satisfies $8 \leq k \leq \min(n - 2\gamma, n(1 - 2/e))$, where γ is the target security level. The default is to set k to be the largest value allowed by this inequality, rounded down to a multiple of 8.

The introduction to MS DRBG (section C.4.3.1) applies very different output security bounds to RSA PRG than MS DRBG: it states that the $k = \lg \lg N$ least significant bits the RSA generator outputs "are (asymptotically in N) known to be as secure as the RSA function f. The Micali-Schnorr generator MS_DRBG() uses the same e and N to produce many more random bits per iteration, while eliminating the reuse of bits as both output and seed."

Modulus and exponent generation. The ISO standard states that implementations "shall" permit either an implementation-generated "private modulus" or the use of one of the default moduli in the standard (see Figure 1.) The standard requires the length of the modulus to conform to the requested security strength: a 1024-bit N for $\gamma = 80$; 2048 bits for $\gamma = 112$; 3072 for $\gamma = 128$; 7680 for $\gamma = 192$; and 15360 for $\gamma = 256$.

The standard also specifies that custom RSA moduli should be generated so that $p - 1$, $p + 1$, $q - 1$, and $q + 1$ have a prime factor of at least γ bits. It also states that the default moduli have been generated so that $p = 2p_1 + 1, q = 2q_1 + 1$ for p_1, q_1 primes, and that $p + 1$ and $q + 1$ have "the required large prime factor".

It is also required that e be relatively prime to $(p-1)(q-1)$. While this is necessary for these to be well-defined RSA parameters, the decryption exponent $d = e^{-1} \bmod (p-1)(q-1)$ is never used in the normal course of random number generation, so it is not clear why this requirement needs to be present.

The default moduli are stated to have "strong" primes as factors, which "essentially guarantees" that $\varphi(N)$ will be relatively prime to odd e, but the factorization of these default moduli is not given, so users are unable to verify this for themselves when using the default moduli with user-generated e. If a user-generated exponent is not supplied, the default $e = 3$ is used.

Backtracking and prediction resistance. The standard states that "[backtracking resistance] is inherent in the algorithm, even if the internal state is compromised." This is not true against an adversary who knows the factorization of the modulus: if the state is compromised, the adversary can distinguish a sequence of previous outputs from random by iteratively decrypting. The standard ensures prediction resistance by requiring the implementation to reseed every 50,000 outputs.

2.2.3 ANSI X9.82

The Dual EC and MS DRBG algorithms were the two number-theoretic (public-key cryptographic) PRG designs promoted by the NSA for inclusion in ANSI X9.82 [Joh04]. A version of MS DRBG was present in early drafts of the ANSI X9.82 specification from 2004 until August 2005, when it was removed.

The text of the entire X9.82 DRBG specification is largely identical to the standard ultimately published by ISO in 2005 [Int11]. A number of draft versions of the X9.82 standard as well as internal discussions and documentation have been made available as part of a FOIA request from NIST in 2014 and 2015 [Nat15], which provide interesting insights into the development of the standard. There is evidence that this text was written by the US government [BLN16].

There appear to have been differing views among committee members on the number of bits that should be output from MS PRG. Early proposals suggested outputting far fewer bits than the ISO version ultimately standardized. A set of 2004 slides from the NSA at a NIST workshop suggested outputting only the "hardcore" bits for each modulus size; for a 1024-bit RSA modulus the suggestion was 10 bits, and for 2048 and 3072-bit moduli the suggestion was 11 bits [Joh04].

An undated (but apparently early) draft of X9.82 includes the comments "The MS generator allows a much larger percentage of N bits to be used on each iteration, and has an additional advantage that no output bits are used to propagate the sequence. (It does, however, rely on a stronger assumption for its security than the intractability of integer factorization.) As the X9.82 standard evolved, committee members argued for restricting the number of bits generated on each exponentiation to $O(\lg \lg N)$ *hard* bits, as is done in Blum-Blum-Shub. The result is that the efficiency argument for choosing MS over BBS doesn't apply. Nonetheless, a user does have more options in the choice of parameters" [har]. Later drafts of the text mention only the larger output lengths ultimately adopted by ISO.

The sole comments we have located that justify the decision to drop Micali-Schnorr from the standard come from a document titled "DRBG recommendations from the X9.82 Editing Group" which states "We recommend keeping DUAL_EC_DRBG. Despite the fact that it is much slower than the other DRBGs, it offers a third distinct technology that can serve as a hedge against breakthroughs in cryptanalysis of hashes and block ciphers. ... We suggest dropping HASH_DRBG and the MS_DRBG, as well as support for the other NIST curves in the DUAL_EC_DRBG" [drb].

2.3 Related work

Backdoored random number generation. In addition to works cited in the introduction, a long line of literature considers the possibility of *algorithm substitution attacks* (previously referred to as "subversion" and "kleptographic") attacks [YY97, YY96, BPR14, RTYZ16, BL17, FM18, CRT+19, PW20]. To formalize this work into the setting of PRGs, Dodis et al. [DGG+15] give a formal treatment and prove that such schemes are equivalent to public-key encryption schemes with pseudorandom ciphertexts. Degabriele et al. [DPSW16] extend these results to consider backdoored PRNGs (which unlike PRGs may take additional inputs for prediction resistance).

Backdoored RSA parameters. There is a surprisingly long line of work on generating "backdoored" RSA parameters [And93, YY96, YY97, YY06, CS03, Joy08, Pat12, WKM16, Ces22]. This work focuses on trapdoors that admit efficient factorization or recovery of private keys given only a public key (N, e). In contrast to these works, our work begins from the assumption that the adversary possesses the factorization of N and addresses the problem of compromising an algorithm using this knowledge.

MS DRBG. Additionally, some previous works have considered MS DRBG. Fouque, Vergnaud, and Zapalowicz give a time/memory tradeoff for recovering the state faster than brute force [FVZ13], and Fouque and Zapalowicz study the statistical distance of short RSA [FZ14]. In a 2013 blog post Matthew Green posed the problem of finding a practical attack against MS DRBG when the factors are known [Gre13]. Antonio Sanso suggested in a 2017 blog post that Mersenne or other special-form primes might lead to a backdoor in Micali-Schnorr [San17]. Lynn Engelberts extended Sanso's analysis of special form primes in a 2020 masters thesis. [Eng20]

3 Security Reductions for the MS and RSA PRGs

The security reduction that Micali and Schnorr give to their "sequential" construction has two steps. First, they define a security question, which they label **Q1**. We rephrase this as an assumption below:

Assumption 1 ([MS91])**. (Q1)**. *The following distributions are polynomially indistinguishable*

- $(N, s^e \bmod N)$ *for* $s \in_R [1, N2^{-k}]$
- (N, r) *for* $r \in_R [1, N]$.

The authors next provide a polynomial-time reduction that transforms a distinguisher algorithm for the Micali-Schnorr PRG into a distinguisher for Assumption 1. The proof uses a hybrid argument. It is this assumption that is used to justify the large output sizes used in the ISO version of MS PRG. The second half of the reduction completes the reduction to the hardness of RSA inversion, and gives a security reduction for both MS PRG and RSA PRG.

Theorem 1 ([MS91])**.** *Let N be an RSA modulus. Every probabilistic algorithm that ϵ-rejects ciphertexts of random messages $s \in_R [1, N2^{-k}]$ can be transformed into a probabilistic algorithm for decrypting arbitrary RSA ciphertexts; this algorithm terminates after at most $(2^k \epsilon^{-1} n)^{O(1)}$ steps.*

This reduction contains several steps that are fundamentally exponential time in k, the number of bits of output. In particular, the algorithm samples 2^k messages until it expects to find one with the required number of zeros. Thus, this reduction is only polynomial time for $\lg n$ bits of output. Fischlin and Schnorr [FS00] have improved the running time of the ACGS algorithm [ACGS84] used in the decryption step, but the reduction remains exponential in k.

In addition to being exponential time in the output length k, running the reduction will be more expensive than simply brute forcing the unknown bits of the plaintext when $k > n/2$, that is, when the output is larger than the state. In fact, the constants hidden in the $O(1)$ result in significantly worse parameters.

The exponential cost in the reduction in the proof of Theorem 1 appears to be the reason for only outputting $\lg n$ bits of output for the RSA PRG. It is interesting that the ISO standard accepts much more generous parameters for Micali-Schnorr output than for the RSA PRG without having attempted to find an analogously relaxed assumption that might permit more generous outputs as Steinfeld, Pieprzyk, and Wang [SPW06] ultimately did.

Statistical indistinguishability results and mod p variants. Micali and Schnorr also consider a variant of their PRG defined modulo a prime p, and hypothesize that this variant is still secure despite the fact that their factoring-based assumptions no longer hold. The journal version of their paper [MS91] contains theorems proving the statistical randomness of the $(n/2-k-(\lg n)^2)$ least significant bits of $s^e \bmod N$ for $s \in_R [1, N^{2^{-k}}]$ when N is prime or an RSA modulus.

Fouque and Zapalowicz [FZ14] give a more general version of this theorem for RSA moduli that applies to size bounds above $\sqrt{N}$ and prove that the $\lg N$ least significant bits of $s^e \bmod N$ for $s < M$ for a chosen bound $M < N$ are statistically indistinguishable from uniform. They apply their bounds to Micali-Schnorr and find that asymptotically, this bound dictates that the output is not statistically indistinguishable when more than $n/3$ bits are output.

These statistical indistinguishability results provide evidence that least significant bits of modular exponentiation (modulo primes or RSA moduli) are indistinguishable from uniform at much less aggressive parameters than ISO chose. However, such statistical indistinguishability results cannot apply when the output exceeds the length of the seed. Thus they do not rule out the possibility of attacks on the PRG with long or multiple outputs.

4 Ruling out black-box attacks

In this section, we will try to make more precise the intuition that the security of MS and RSA PRG is more closely related to the assumption of pseudorandomness of RSA ciphertexts than the hardness of inverting RSA. This offers a more formal explanation for why there does not appear to be a black-box way to use an RSA decryption oracle to break the security of MS PRG.

We begin by defining a generic Micali-Schnorr-type construction, which we call MS-f-PRG. In this variant the RSA operation is replaced by some function f. That is, Step 3 of Algorithm 2 in Section 2.2.1 becomes $z_i \leftarrow f(s_{i-1})$, for f that we will instantiate below.

4.1 Micali-Schnorr is secure with a PRG

The Micali-Schnorr construction is still secure when instantiated with a PRG.

Theorem 2. *If $f : [1, 2^{n-k}] \to [1, 2^n]$ is a secure pseudorandom generator, then the output of MS-f-PRG is pseudorandom.*

The proof is the same as the proof of Theorem 5.1 in [MS91], substituting the pseudorandomness of G for Assumption 1.

While this finding is not surprising, it illustrates that the security of MS-f-PRG need not depend on any secret information. This informs how a provably secure variant of MS PRG could be instatiated, but unfortunately it does not enable a proof for the variants with RSA modulus N or prime modulus p (as discussed in the original work of Micali and Schnorr [MS91]). In the context of an adversary who knows the factorization of N, $G(s) = s^e \bmod N$ with short seed s is distinguishable from random: simply decrypt $G(s)$ and check if the seed is short. This function is clearly not a PRG, and so we gain no information about the security of this construction. A similar argument applies in the prime modulus case $f(x) = x^e \bmod p$.

4.2 MS PRG is still secure when implemented with a random permutation

We next show that the Micali-Schnorr construction is secure when the one-way RSA function is replaced with a public (invertible) random permutation. While this analysis is clearly quite artificial, it offers a useful bound on the efficiency of *generic attacks* (i.e., attacks that do not exploit special properties of the RSA function) when the factorization of N is known.[5] This suggests that if it is indeed possible to backdoor the Micali-Schnorr construction, the backdoor must take advantage of some nontrivial algebraic property of RSA.

We begin by defining our variant of the MS PRG in which RSA encryption is replaced with a publicly accessible random permutation. Concretely, at line 3 of Algorithm 2 we replace the RSA evaluation $z_i \leftarrow s_{i-1}^e \bmod N$ with $z_i \leftarrow f(s_{i-1})$ where $f : [1, N-1] \to [1, N-1]$ is a publicly accessible random permutation. We give the attacker the ability to decrypt by making the inverse permutation f^{-1} publicly accessible. With this modification, we obtain the following theorem.

Theorem 3. *No adversary $\mathcal{A}$ that makes q total queries to black-box oracles for random permutations f and f^{-1} can distinguish the hk-bit output string B of our modified variant of Algorithm 2 from a random hk-bit string with advantage greater than $\frac{(h+1)^2}{2N} + \frac{2hq}{N2^{-k}} + \frac{hq}{N} + \frac{2hq}{N-q} + \varepsilon$ (for some negligible ε).*

A proof of Theorem 3 can be found in Appendix A.1.

4.3 RSA-PRG as a sponge

The iterative construction of RSA-PRG—apply a transformation to the state, then output the LSBs—is widely used in symmetric cryptography, and is known as the sponge construction. If we replace RSA encryption with a random function f in the RSA PRG construction, we can use theorems developed for cryptographic sponge constructions to obtain strong bounds on the security of the resulting construction. These theorems hold for functions that are fixed public and efficiently invertible permutations.

[5] More critically, this construction does not have any implications for the security of MS PRG instantiated with the RSA function, since RSA encryption quite clearly behaves differently than a random function.

Theorem 4. *Let f-PRG be the RSA PRG construction except replacing the $x \mapsto x^e \bmod N$ operation with a fixed, public, efficiently invertible random permutation $f : \mathbb{Z}_N \to \mathbb{Z}_N$. Then the output of f-PRG is indistinguishable from random to any adversary that runs in time polynomial in $\lg N - k$ and has black-box access to f and f^{-1}.*

Proof. f-PRG follows the sponge construction, with state size $\lg N$, rate k, and capacity $c = \lg N - k$. As Bertoni, Daemen, Peeters, and Van Assche show in equation 6 of [BDPV08], the $\mathcal{RO}$ differentiating advantage against the sponge construction when used with a random permutation is upper bounded by $m^2 2^{-(c+1)}$, where m is the number of calls to the underlying transformation. The $\mathcal{RO}$-differentiation game models an adversary with query access to f and f^{-1}. Thus no $poly(\lg N - k)$-time adversary can distinguish f-PRG from a random oracle with more than negligible probability, even with query access to f and f^{-1}. □

We remark that SHAKE-128 (from the SHA-3 standard [sha15]) is a PRG constructed as a sponge with state size 1600, outputting 5/6 of the state (1344 bits) per iteration, and the function f it uses to transform its state is a fixed, public, efficiently invertible permutation. This is a smaller state size than ISO MS DRBG uses ($\lg N = 3072$ for 128-bit security), and a larger fraction of bits that are output (5/6, compared to 1/3 for ISO using the default exponent $e = 3$).

5 Algebraic Attacks

In this section, we present several attacks against RSA PRG and MS PRG. While none of these results in a backdoor against MS DRBG as standardized that is as compelling as the techniques demonstrated against Dual EC, these attacks illustrate different properties of the algebraic structure of modular exponentiation that may ultimately lead to either the development or ruling out of such a backdoor in the MS PRG.

5.1 Notions of cryptographic subversion

A growing body of work considers *algorithm substitution attacks* (ASAs) against cryptographic implementations. In this setting, a known cryptographic algorithm is replaced with a subverted algorithm designed by an attacker, who retains secret knowledge that allows for exploitation [YY97, YY96, BPR14, RTYZ16, BL17, FM18, CRT+19, PW20]. This effectively models our assumptions for a subversion attack on MS DRBG. We do not present formal definitions here, and refer the reader to e.g., [BPR14, RTYZ16] for details.

The MALICIOUS framework. Peyrin and Wang [PW20] describe the MALICIOUS framework that includes several informal properties required by a subverted symmetric construction such as an RNG. We modify these to provide a shorthand characterization for our backdoor constructions.

Undiscoverability: An outside observer should be unable to find the hidden backdoor, even if the general form of the backdoor is known [PW20].

Practical Construction: The backdoor designer should be able to efficiently construct parameters that allow for backdoor exploitation.

Practical Exploitation: An attacker should be able to efficiently violate the security properties of the scheme if they know the secret information required to exploit the backdoor.

Plausible Deniability: To an external observer, the public parameters, keys, and structure of the cryptosystem should appear to be "properly" generated.

Relationship to formal definitions of ASAs. Bellare, Paterson and Rogaway formalized a framework for *algorithm substitution attacks* (ASAs) [BPR14]. Informally, the framework captures the properties described above using two independent security games.[6] In the first game, a *detection adversary* ($\mathcal{D}$) represents the defender, and is used to define the undiscoverability of a cryptographic backdoor. In this framework, the detection adversary is asked to distinguish between the correct algorithm (e.g., an implementation of MS DRBG in which the modulus N is generated honestly at random and no secret factorization is retained) and a second *subverted* algorithm such as the ISO standard with attacker-known (or chosen) factorization. The subverted algorithm passes this test if $\forall$ P.P.T. algorithms $\mathcal{D}$, $\mathcal{D}$ distinguishes the input/output behavior of the subverted implementation from the correct implementation with at most negligible advantage.[7]

In the second game a *subversion adversary* ($\mathcal{S}$) is given access to secret knowledge about the subverted algorithm (for example, knowledge of the secret factorization of the modulus N.) The minimal criteria for a subversion attack is that there must exist some P.P.T. $\mathcal{S}$ that distinguishes the input/output behavior of the subverted and non-subverted implementations with non-negligible advantage. If the non-subverted implementation is itself a secure PRG, then the ability to distinguish between subverted and unsubverted implementations naturally implies an attack that distinguishes the output of the subverted algorithm from random bits. In practice, subversion attackers may also be able to carry out more powerful attacks, such as state recovery and future output prediction.

5.2 Algorithmic Background: Multivariate Coppersmith's Method

Several of our attacks make use of the following version of multivariate Coppersmith's method. For a basic review of lattices and terminology, see [LLL82].

5.2.1 Review of Coppersmith's method.

Coppersmith's method uses lattice reduction to find small solutions to polynomials modulo integers. For univariate polynomials, this method is fully rigorous, and has a clean bound: for a degree-d polynomial $f(x) \in \mathbb{Z}[x]$ and $N \in \mathbb{Z}$, all roots $r \in \mathbb{Z}$ satisfying $f(r) \equiv 0 \bmod N$ can be found for $|r| < N^{1/d}$ in polynomial time in d and $\lg N$ [Cop97].

The multivariate generalization of this method that we need for our attacks does not have a clean theorem statement, and in fact a fully rigorous generalization cannot exist [Cop01]. Nevertheless, a heuristic generalization of this method often works in practice [Jut98], and it is this heuristic version that we will use. We will derive the relevant bounds using ad hoc, problem-specific constructions.

The following lemma tells us the condition under which we expect to succeed.

[6] The "Practical Construction" requirement is captured formally by requiring that both honest and subverted algorithm can have a polynomial-time *setup* algorithm that (in the subverted case) produces the subverted implementation and any secret trapdoors.

[7] Subsequent definitions by Russel *et al.* extend this notion to one in which the detection adversary (in this work called an "online watchdog") is also allowed to observe interactions between the implementation and an attacker. We do not consider this scenario in our work, due to the fact that we primarily focus on passive eavesdropping attacks.

Lemma 1. *Let $\{f_i(\vec{x})\}_{i=1}^{w}$ be integer polynomials in m variables $\vec{x} = (x_1, \ldots, x_m)$ and let $N \in \mathbb{Z}$. We wish to find one or more solutions $\vec{r} = (r_1, \ldots, r_m)$ simultaneously satisfying $\{f_i(\vec{r}) \equiv 0 \bmod N\}_{i=1}^{w}$.*

If we can find m auxiliary polynomials $Q_1, \ldots, Q_m$ such that

$$Q_j(r_1, \ldots, r_m) \equiv 0 \bmod N^t \qquad \text{and} \qquad |Q_j(r_1, \ldots, r_m)| < N^t$$

for some integer $t \geq 1$ then each Q_j satisfies $Q_j(r_1, \ldots, r_m) = 0$ over the integers. If in addition the Q_j are algebraically independent, then we can solve for a bounded number of possible solutions.

We sketch a general method to solve this problem in Algorithm 3 below.

Algorithm 3: Multivariate Coppersmith Method (Sketch)

Input : $\{f_i(\vec{x})\}_{i=1}^{w} \in \mathbb{Z}[x_1, \ldots, x_m]^w, N \in \mathbb{Z}, \{R_j\}_{j=1}^{m}$
Output: $\{r_j\}_{j=1}^{m}$ satisfying $|r_j| < R_j$ and $f_i(\vec{r}) \equiv 0 \bmod N$

1. Generate a basis of auxiliary polynomials of the form $g_{\vec{a},\vec{b}}(\vec{x}) = \left(\prod_j x_j^{a_j}\right)\left(\prod_i f_i^{b_i}\right) N^{t-\sum_i b_i}$.
2. Map each polynomial to a scaled coefficient vector embedding: $\sigma : g(\vec{x}) = \sum_{\vec{c}} g_i x_1^{c_1} x_2^{c_2} \ldots x_m^{c_m} \mapsto (g_1 R_1^{c_1} \ldots R_m^{c_m}, \ldots,)$
3. Construct a lattice basis B of coefficient vector embeddings $\sigma(g)$ for a carefully chosen subset of the gs generated in step 1.
4. LLL-reduce the lattice basis.
5. Construct a Gröbner basis of the polynomials $\sigma^{-1}(v)$ of all vectors v in the reduced basis whose ℓ_1 norms $|v|_1$ are shorter than N^t.
6. Enumerate the candidate solutions given by the Gröbner basis and verify whether each is a valid solution for the r_i.

The value t and choice of polynomial shifts $x_1^{a_1} \ldots x_m^{a_m}$ in Step 1 of Algorithm 3 are chosen as part of the optimization process. We will refer to t as the *multiplicity* of the roots. The lattice dimension is determined by the number of distinct monomials in the set of polynomials $\{g\}$ used to generate the lattice.

To apply Lemma 1 we bound $|g(\vec{r})| < |\sigma(g)|_1$, so we want to find m vectors in the lattice whose ℓ_1 norms are less than N^t. For a random lattice L, the successive minima $\lambda_i(L)$ often have close to the same length, and in practice LLL [LLL82] typically finds vectors of length $1.02^{\dim L}(\det L)^{1/\dim L}$ or $1.02^{\dim L}\lambda$ [NS06].

These vectors are guaranteed to be linearly independent as coefficient vectors, but the corresponding polynomials are not guaranteed to be algebraically independent. Nevertheless, the polynomials found by this algorithm for random problem instances with optimal parameters are often algebraically independent.

Thus heuristically we expect this algorithm to succeed if we can construct a lattice basis with determinant bounded as in Condition 1.

Condition 1 (Heuristic Condition for Multivariate Coppersmith)**.** *If the basis for lattice L constructed in Step 3 of Algorithm 3 satisfies*

$$1.02^{\dim L}(\det L)^{1/\dim L} < N^t$$

then we heuristically expect Algorithm 3 to find all suitable roots.

5.2.2 Applying Coppersmith's Method to MS and RSA PRG.

It is tempting to try to apply a multivariate Coppersmith approach directly to MS or RSA PRG to carry out a state recovery attack. In particular, such an attack involves finding a small solution of a degree-e polynomial modulo N, which is precisely the problem that Coppersmith-type methods solve.

In this section, we will sketch this attack and observe that it is ruled out by the parameter choices made by ISO for MS.

MS PRG. An attempted state recovery attack from two outputs would start from the polynomial relations between the unknown states s_i:

$$s_0^e - 2^k s_1 - b_1 \equiv 0 \bmod N$$
$$s_1^e - 2^k s_2 - b_2 \equiv 0 \bmod N$$

Let $|s_i| < R$. Construct the lattice basis

$$B = \begin{bmatrix} R^e & 0 & -2^k R & 0 & -b_1 \\ 0 & R^e & 0 & -2^k R & -b_2 \\ 0 & 0 & NR & 0 & 0 \\ 0 & 0 & 0 & NR & 0 \\ 0 & 0 & 0 & 0 & N \end{bmatrix}$$

We have $\det L(B) = R^{2e+2}N^3$ and $\dim L(B) = 5$. Omitting approximation factors in such small dimension, Condition 1 tells us we expect to succeed if

$$(\det L(B))^{1/\dim L(b)} = (R^{2e+2}N^3)^{1/5} < N$$

which applies when $R < N^{1/(e+1)}$. In other words, the bit length of the state size $r = n - k$ should satisfy $r < n/(e+1)$. Attempted improvements from higher degree polynomials and root multiplicities seem to give the same bound, even if more than two outputs are available.

This attack is ruled out by the choice of ISO parameters $r = 2n/e$. This makes sense because this attack does not even require the factorization of N.

RSA PRG. Steinfeld, Pieprzyk, and Wang [SPW06] do a similar analysis of RSA PRG and obtain a heuristic bound of $r < n/(e+1)$ for the unknown portion of the state. Herrmann and May [HM09] improve this to n/e when the PRG outputs the most significant bits of the state.

Simpler attacks when $r < n/e$. A state size bound of $r < n/e$ is a degenerate case for both RSA and MS PRGs, since there is no modular reduction performed when computing $s_i^e \bmod N$. Fouque, Vergnaud, and Zapalowicz point out that one can recover the state via Hensel lifting [FVZ13].

5.3 Attacks on RSA PRG

In this section, we show how to construct backdoor parameters for the RSA PRG. The states (and thus the output) generated by the RSA PRG have an iterative structure that cycles modulo a divisor of $\varphi(\varphi(N))$. This means that an attacker who can control the generation of N and e can embed a chosen relationship among outputs that enables efficient distinguishing and state recovery attacks.

5.3.1 e has Short Period (eSP) attack mod $\varphi(\varphi(N))$

In our first backdoor construction, we show that it is possible to efficiently construct RSA parameters for which the output of RSA PRG produces extremely short cycles. This violates indistinguishability, but would be observable by any attacker. We then show that it is possible to somewhat obscure the most obvious cyclic behavior in the output, which leads to an efficiently generatable and exploitable backdoor. While this behavior alone doesn't lead to a fully undiscoverable backdoor, it provides intuition for the SUS backdoor we construct later.

Recall that the multiplicative order of an integer modulo N is a divisor of $\varphi(N)$. In the RSA-PRG generator with modulus N and exponent e, we have state s_i satisfying $s_i = s_0^{e^i} \bmod N$. Micali and Schnorr (as well as Blum, Blum, and Shub [BBS86]) note that the period of the sequence of outputs generated by s_0 will thus be a divisor of $\varphi(\varphi(N))$ [MS91]. They say that "in general" the period "will be a large factor of $\varphi(\varphi(N))$ and will be much larger than $\sqrt{N}$ which is the average period of a random recursion in $\mathbb{Z}_N$. It is conceivable that the number $\varphi(\varphi(N))$ somewhat affects the output distribution of the generator and not only its period." They do not appear to have considered the possibility of malicious parameter generation.

We consider three cases:

Case 1: e not relatively prime to $\varphi(N)$. If (in violation to valid RSA parameters), e is not relatively prime to $\varphi(N)$, then the $x \mapsto x^e \bmod N$ operation is not surjective—not every element mod N is an eth power. Suppose $\varphi(N) = we^j$ for some w. Then the jth PRG state $s_j = s_0^{e^j}$ will be a wth root of unity mod N, and so will all subsequent PRG states. Thus:

$$s_{j+i} = s_j^{e^i \bmod w},$$

giving a cycle whose period is the multiplicative order of e mod w. If w is small, or if e generates a small subgroup of $\mathbb{Z}^*_{\varphi(N)}$, then this will lead to short cycles.

We note that for "default" RSA moduli that are given without factorization, there is no way for the user to efficiently verify that a user-chosen e is relatively prime to $\varphi(N)$. However, this construction is probably best considered to be not plausibly deniable, since these are not valid RSA parameters.

Case 2: cycles with e relatively prime to $\varphi(N)$. Alternatively, even if e is relatively prime to $\varphi(N)$, e could potentially generate a small subgroup modulo $\varphi(N)$, leading to this type of short cycle. One algorithm, given a specific e, to find primes p such that e has small order modulo $\varphi(p)$ is shown in Algorithm 4.

This algorithm is reasonably efficient in practice for parameters of interest. For example, for $e = 5$ and $\ell = 504$, it took 10 seconds on a laptop with a dual-core Intel i7-6500 CPU to find an 880-bit prime with these properties using Sage with ECM for factorization, aborting factorization when it started to get slow. We tried a few candidates for e and ℓ within this range. To generate a hard-to-factor modulus N, one could generate two primes using this algorithm.

These types of cycles in the output would be easy to notice for any user who generates ℓ outputs, so this construction would be discoverable.

Case 3: Partially hidden cycling behavior. Here we will outline the most practical eSP (e has Short Period) backdoor variant, in which we hide the obvious cycling behavior in the factorization

Algorithm 4: Constructing prime p s.t. e has small order mod $\varphi(p)$.

Input : An integer e

Output: A prime p such that e has small order modulo $\varphi(p)$.

1 Choose a cycle length ℓ.

2 Compute small prime factors p_i of $e^\ell - 1$ using the elliptic curve method or other factoring methods that are efficient for small factors.

3 Choose a subset of the p_i (and optionally also the composite cofactor) computed in the previous step, and check if $1 + \prod_i p_i$ is prime. (In order to generate odd primes, we will need one of the p_i to be 2.)

of $N = pq$. If e generates cycles of length ℓ mod p, but not mod q, the outputs would not have as obvious cycling behavior to the end user. By choosing q such that $e^\ell \equiv c_q \bmod \varphi(q)$ for some c_q small enough that finding roots mod q of degree c_q polynomials is feasible, an adversary who knows the factorization and observes the sequence of outputs can efficiently recover the full state, as we will show in Theorem 5. Such a q can be generated similarly to Algorithm 4, but factoring $e^\ell - c_q$ instead of $e^\ell - 1$.

With such parameters, an attacker can recover the full state from the first $(\ell+1)$ PRG outputs (and in particular, using only the first and $(\ell+1)$th outputs) assuming each output has length $k \geq n/2$. (If RSA PRG parameters were set following the ISO parameters for MS DRBG, this will be the case for all $e \geq 5$.) The attack is given in Algorithm 5.

Algorithm 5: eSP attack in the "partially hidden cycle" case.

1 Let b_1 and $b_{\ell+1}$ be two outputs. Without loss of generality assume $s_{\ell+1} \geq s_1$ as integers.

2 Observe that $b_{\ell+1} - b_1 \equiv (s_{\ell+1} - s_1) \bmod 2^k$. Observe further that $s_{\ell+1} - s_1 \equiv 0 \bmod p$, and $0 \leq (s_{\ell+1} - s_1)/p < q < 2^k$.

3 Let $m = (b_{\ell+1} - b_1)p^{-1} \bmod 2^k$, reduced such that $0 \leq m < 2^k$. Now $m = (s_{\ell+1} - s_1)/p$ as integers.

4 Solve the polynomial congruence

$$y^{c_q} - y - pm \equiv 0 \pmod{q}.$$

(Recall $c_q \equiv e^\ell \bmod \varphi(q)$.) This is feasible because the degree c_q is small and q is prime.

5 One of the roots will be s_1 mod q, because mod q we have $s_1^{c_q} \equiv s_1^{e^\ell} \equiv s_{\ell+1} \equiv pm + s_1$.

6 For each root α, use CRT to recover $\tilde{s_1} \in [0, 2^k q)$ such that $\tilde{s_1} \equiv b_1 \pmod{2^k}$ and $\tilde{s_1} \equiv \alpha \pmod{q}$, and check whether $\tilde{s_1}$ would produce correct outputs b_i.

7 Since $s_1 < pq < 2^k q$, when $\alpha = s_1 \bmod q$, we will get $\tilde{s_1} = s_1$ over the integers.

As a proof of concept, we generate (for public exponent $e = 5$) a 2048-bit backdoored $N = pq$ such that $5^{504} \equiv 1 \pmod{p-1}$ and $5^{504} \equiv 187 \pmod{q-1}$. We include it in Appendix A.2. The state recovery attack (implemented in Sage) using this modulus took 31 seconds.

However, while practical to construct and practical to exploit, this "backdoor" is not undiscoverable (regardless of how q is generated) because a user could exploit the relationships modulo p to efficiently factor N as follows. They choose an initial state s_0, and compute the sequence of states $s_i = s_{i-1}^e \bmod N$. If they discover a state s_ℓ where $\gcd(s_\ell - s_0, N)$ is nontrivial, then they

can use this to factor N. (Note that this algorithm is similar to the Pollard rho algorithm but with a different "pseudorandom" walk.)

We summarize these results in the following informal theorem:

Theorem 5. *An attacker can efficiently generate RSA parameters $(N = pq, e)$ such that e has a chosen (small) order $\ell > \log_e \varphi(p)$ modulo $\varphi(p)$. This attacker can then carry out an efficient state recovery attack after observing at least $\ell + 1$ RSA PRG outputs of length $k \geq n/2$ bits generated using these parameters. This backdoor has efficient parameter generation and efficient exploitation but is discoverable.*

While these ideas do not generate a fully satisfactory backdoor, they provide intuition for the construction in the next section, where we will replace the relationship $e^\ell \equiv 1 \bmod \varphi(N)$ (or $\bmod \varphi(p)$) with a more complex polynomial.

5.3.2 The SUS backdoor for RSA-PRG

In order to conceal the discoverable cyclic behavior of the eSP backdoor, we augment this idea to generate moduli that embed a small, sparse polynomial relationship satisfied by the exponent e modulo $\varphi(N)$ or $\varphi(p)$. We will call this "Small Unknown Solution" or SUS.

Parameter Generation. Let $f(x) = \sum_{i \in S} c_i x^i$ be a sparse polynomial that will remain secret, where the c_i are all ± 1 and are roughly balanced. A correspondingly backdoored prime p will satisfy the relation $f(e) \equiv 0 \bmod p - 1$. This corresponds to a relation between PRG states $\prod_{i \in S} s_i^{c_i} \equiv 1 \pmod p$, where $s_i = s_0^{e^i}$.

To backdoor an RSA modulus N, we can either ensure $f(e) \equiv 0 \bmod \varphi(N)$, or (similarly to the "partially hidden cycle" eSP variant) have $f(e) \equiv 0 \bmod \varphi(p)$ but not $\bmod \varphi(q)$.

Theorem 6. *SUS prime and RSA modulus generation is efficient.*

Proof. We apply Algorithm 4 except that we replace the desired relation $e^\ell - 1 \bmod p - 1$ with a sparse polynomial $f(e) = \sum_i c_i e^i$. To generate an RSA modulus, we can either generate two primes from different subsets of the factors in Step 3 (so that $f(e) \equiv 0 \bmod \varphi(N)$), or we can backdoor p and choose q normally (so that $f(e) \equiv 0 \bmod \varphi(p)$). □

SUS State recovery attack. The state recovery algorithm uses multivariate Coppersmith. We write $s_0^{e^i} = s_i = b_i + 2^k r_i$, using the outputs b_i and unknown state MSBs r_i, with $0 \leq r_i < R = N/2^k$. For ease of exposition let us assume for now that $f(e) \equiv 0 \bmod \varphi(N)$; applying our backdoor polynomial f we obtain

$$\prod_{i \in S} \left(2^k r_i + b_i\right)^{c_i} \equiv 1 \bmod N$$

This is a low-degree multivariate polynomial modulo N whose roots are the unknown portions of each state. Recall all c_i are ± 1, with roughly balanced sets S^+ of positive c_i and S^- of negative c_i. This gives

$$\prod_{i \in S^+} (2^k r_i + b_i) - \prod_{i \in S^-} (2^k r_i + b_i) \equiv 0 \bmod N$$

Our polynomial degree is $\max(|S^+|, |S^-|)$, which is independent of e. We can then recover the r_i using multivariate Coppersmith.

If instead we had $f(e) \equiv 0 \bmod \varphi(p)$ but not $\bmod \varphi(q)$, we would instead recover the $r_i \bmod p$. But as long as $p > R$ this is the same as recovering r_i over the integers.

Example. Suppose $f(e) = e^{200} + e^{20} - e^{180} - e^0 \equiv 0 \pmod{\varphi(N)}$; we have $|S^+| = |S^-| = 2$. The Coppersmith polynomial in unknowns $r_{200}, r_{20}, r_{180}, r_0$ is

$$f(\vec{s}) = (r_{200} + 2^{-k}b_{200})(r_{20} + 2^{-k}b_{20}) - (r_{180} + 2^{-k}b_{180})(r_0 + 2^{-k}b_0).$$

We apply Algorithm 3 with $t = 1$ and no extra shifts to generate a lattice with $\dim L = 7$ and $\det L = R^8N^6$ (where R is our bound on the r_i). Applying Condition 1 (and omitting the approximation factor in dimension 7), we expect to succeed when $(R^8N^6)^{1/7} < N$, or when $R < N^{1/8}$. Had we instead had $f(e) \equiv 0 \pmod{\varphi}(p)$, our success condition would instead be that $R < p^{1/8}$. In either case, this bound is independent of e.

As a demonstration, we generated a 1024-bit RSA modulus N satisfying $e^{200} - e^{180} + e^{20} - 1 \equiv 0 \bmod \varphi(N)$ for $e = 17$. We include it in Section A.3 of the supplementary materials. Parameter generation took 19 seconds using Sage on a single core of a machine with an Intel E5-2699 processor. Using these parameters with $k = 896$-bit outputs, our attack successfully recovered the state in 213 milliseconds from 200 PRG outputs. For these parameters, the fraction of bits output is below the $(1 - 1/e)$ fraction required by Herrmann and May [HM09]; that is, our attack requires less output to succeed than theirs. In fact, the fraction of bits output is smaller than $(1 - 2/e)$, the maximum fraction of output bits recommended in the ISO parameters for Micali-Schnorr — although when N is 1024 bits (at the 80-bit security level) the ISO standard recommends only 864 bit outputs, to ensure at least 160 bits remain unknown. In practice, however, LLL reduction of this Coppersmith lattice yields shorter vectors than predicted, and our attack empirically succeeds for these example parameters with ISO-sized $k = 864$ outputs, and even with outputs as small as $k = 856$ bits.

Using a higher multiplicity (and thus a larger-dimension lattice) allows the attack to succeed with even smaller outputs:

Theorem 7. *A SUS-backdoored modulus N of length n with backdoor polynomial f of degree ℓ and $|S|$ nonzero coefficients allows an efficient state recovery attack after observing ℓ outputs of length $k > n(1 - 1/c_S)$ for a constant c_S that depends only on $|S|$, and not on the exponent e.*

When f has $|S| = 2$ terms, $c_S = 2$, when f has $|S| = 4$ terms, $c_S < 6.55$, and when $|S| = 6$ terms, $c_S < 16.96$.

The attack requires only $|S|$ outputs within this range at specified positions.

Proof. The recovery algorithm works as follows. Let $f(x) = \sum_i c_i x^i$. Let $S^+ = \{i \mid c_i > 0\}$ and $S^- = \{i \mid c_i < 0\}$. Apply multivariate Coppersmith's method to solve for the unknown r_i in $\prod_{i \in S^+}(2^k r_i + b_i)^{c_i} - \prod_{i \in S^-}(2^k r_i + b_i)^{|c_i|} \equiv 0 \bmod p$.

For $|S| = 2$, we can construct a full-rank 3-dimensional lattice with multiplicity $t = 1$ to obtain the bound in the theorem. This case is degenerate in that a polynomial with coefficients $+1, -1$ will generate output that cycles. We obtain the bound in the theorem for $|S| = 4$ from a full-rank 1365-dimensional lattice with $t = 8$; for $|S| = 6$, a 1443-dimensional lattice with $t = 4$.

We have chosen these values so that running LLL for these lattices is within feasible range today; one can get improved bounds for the c_S by choosing larger multiplicities and generating larger (but still polynomially sized) lattices. □

Undiscoverability vs. practical exploitation. We hypothesize that SUS-backdoored parameters could be undiscoverable, if the sparse backdoor polynomial f is properly chosen. However, making the backdoor harder to discover seems to make it harder to exploit.

The backdoor polynomial f in the SUS attack is a sparse polynomial with the property that $f(e) \equiv 0 \bmod \varphi(N)$ or $\bmod \varphi(p)$. The more terms in f, the harder it is to guess, but also the higher the degree of the multivariate polynomial to be solved using Coppersmith's method. For Coppersmith's method to succeed this then requires either a larger fraction of bits to be output or a much larger lattice to be reduced.

If f has too few terms, it becomes possible to guess f by brute force, and then verify a guess by checking (for some arbitrary a) whether $a^{f(e)} \equiv 1 \bmod N$ (if $f(e) = 0 \bmod \varphi(N)$) or if $\gcd(a^{f(e)} - 1, N)$ is nontrivial (if $f(e) = 0 \bmod \varphi(p)$).

As an example, suppose we want f to have eight nonzero terms. The multivariate polynomial to be solved using Coppersmith will have degree 4 ($|S^+| = |S^-| = 4$). If we assume the RSA PRG is reseeded every 50000 outputs (as the ISO standard recommends for MS DRBG), the degree of f must be less than 50000, since any outputs after the first 50000 will not be algebraically related to the first 50000 outputs. The size of the search space for f would be roughly $\binom{50000}{7} \approx 2^{97}$. (Without loss of generality we can assume the degree of f is as large as possible, because if $f(e) \equiv 0$ then also $e^{49999-\deg f} f(e) \equiv 0$, so there are only 7 terms to choose.)

However, it may be possible to do better than brute force. If parameters are chosen such that $f(e) \equiv 0 \bmod \varphi(N)$, a meet-in-the-middle attack could recover this f in $\binom{50000}{4} \approx 2^{58}$ time and $\binom{50000}{3} \approx 2^{44}$ space: enumerate over all possibilities f_0 for the first three terms of f, and put $a^{f_0(e)} \bmod N$ into a hashtable. Then iterate through all possibilities for the remaining terms and check for a collision.

We do not see an analogous meet-in-the-middle attack when instead parameters are chosen such that $f(e) \equiv 0 \bmod \varphi(p)$ if p is kept secret. We conjecture that no faster attack than brute-forcing f is possible in this case.

To illustrate the tradeoffs, in addition to the earlier example, we generated two backdoored parameter sets: one requires as high as the 3500th output, took 4 core-hours to exploit, and we conjecture is 2^{51}-undiscoverable; the other requires only as high as the 150th output, took 3 core-minutes to exploit, and is (conjectured) 2^{19}-undiscoverable.[8] We give these (and other) parameters and discuss the tradeoffs further in Appendix A.3.

Extending this idea to Micali-Schnorr. Our attempts to extend this idea from RSA PRG to Micali-Schnorr have encountered some barriers to efficient exploitation that we have been unable to circumvent.

A first barrier is that the output of MS PRG does not follow the clean iterative structure that RSA PRG does. In the case of RSA PRG, this allows us to write the ith block of output b_i as a value that is close to a power of the initial state $s_i \equiv s_{i-1}^e \equiv s_0^{(e^i)} \pmod N$, or a single monomial like x^{e^i} in a polynomial equation we wish to solve. If we attempt to apply the same approach for MS PRG, writing the ith block of output in terms of the initial state by iteratively expanding the expression

$$s_i \equiv 2^{-k}(s_{i-1}^e - b_i) \bmod N,$$

[8] Code demonstrating the attack with these parameters is available at `https://github.com/ucsd-hacc/msdrbg_code`.

this results in a polynomial with exponentially many terms that depend on the previous sequence of outputs.

The minimum degree of our backdoor polynomial $\sum_{i \in S} \pm e^i \equiv 0 \bmod p-1$ needs to be $\log_e \varphi(p)$ in order to embed information modulo p, so our polynomial expression will have exponentially many terms in $\lg p$ to even write down. Using larger coefficients in the polynomial to generate terms like ce^i will increase the degree of the Coppersmith polynomial.

Another way of viewing this obstacle is that the high-degree non-sparse relation between states is due to the simultaneous presence of addition, multiplication, and exponentiation modulo N (or p) in the state update function. If only exponentiation were involved, as is the case of RSA-PRG, we can simplify the expression as above. If only multiplication by a constant and addition were involved, all s_i are affine functions of s_0. When all three operations are involved, however, the resulting expression is a polynomial with exponentially many terms.

One path forward would be to generate some algebraic structure that permits simplification or elimination of enough cross-terms that the polynomial no longer has exponentially many terms and becomes solvable. (We give an example of such a structure in the next section.) Alternatively, we observe that while these polynomials have exponentially many terms if expanded out entirely, they will have linear depth if evaluated as a circuit. Exploiting this idea would require new algorithmic ideas, since a lattice attack requires writing down the polynomial to be solved.

5.4 Attacks on MS PRG

In this section, we give two attacks on weak settings for MS PRG.

5.4.1 Finite Field MS-PRG is insecure.

In this section we define a variant of Micali-Schnorr over finite fields of small characteristic, and detail a straightforward state-recovery attack on this variant that involves no backdoors.

This attack does not imply anything about the existence of an attack (or a feasible backdoor) on standard MS DRBG, but it demonstrates that the pseudorandomness of modular exponentiation depends on the choice of field and illustrates algebraic structure that eliminates the exponential blow-up in terms that kept us from extending the ideas in the SUS backdoor to MS-PRG. This attack works for any choice of output length k, unlike the other attacks we detail.

Finite Field Micali-Schnorr. Define a variant of the Micali-Schnorr PRG using finite fields. Our eventual backdoor will rely on the characteristic of the field matching the exponent e, which we will take to be a small prime.

Let $\mathbb{F}_{e^n}$ be the finite field of size e^n. We can represent elements of $\mathbb{F}_{e^n}$ as polynomials in the quotient ring $\mathbb{F}_e[x]/N(x)$ (with N monic, irreducible, and $\deg(N) = n$) or as coefficient vectors in $(\mathbb{F}_e)^n$. Addition, multiplication, and exponentiation are defined in the standard ways.

Theorem 8 (Informal)**.** *Finite-field Micali-Schnorr with state size n and output size k allows an efficient probabilistic state recovery attack when $\lceil (n-k)/k \rceil$ outputs are observed.*

Attacking FF-MS-PRG. Our attack relies on the linearity of the Frobenius endomorphism to limit the complexity introduced by exponentiation. For any $u, v \in \mathbb{F}_{e^n}$, we have that $(u+v)^e = u^e + v^e$, since all of the cross terms vanish in characteristic e. This linear operation (the Frobenius

Algorithm 6: Finite-Field Micali-Schnorr

Input : Parameters $e \in \mathbb{Z}$, $N \in \mathbb{F}_e[x]$, a number of iterations h
Output: hk output bits

Sample initial state $s_0 \leftarrow \$[1, x^{n-k}]$ using truly random coins.
for $i \leftarrow 1$ **to** h **do**
  $z_i \leftarrow s_{i-1}^e \bmod N(x)$
  Write $z_i = x^k s_i + b_i$ for $\deg(s_i) < n-k$ and $\deg b_i < k$.
end
Output $b_1||b_2||\ldots||b_h$.

endomorphism) is denoted by $F : u \mapsto u^e$. Similarly, the (invertible) action of multiplying a value by x^k is linear and is represented by the map $T : u \mapsto x^k u$.

Using these linear maps, we rewrite the finite field Micali-Schnorr equation $s_{i+1} = x^{-k}(s_i^e - z_{i+1})$ as

$$s_{i+1} = T^{-1}(F(s_i) - z_{i+1}).$$

Clearly this is affine, and thus we may write each s_i as an affine equation in s_0: $s_1 = A_1 s_0 + b_1, s_2 = A_2 s_0 + b_2, \ldots$

When considering these affine equations in the vector space, the constraint $\deg(s_i) < n-k$ ensures that the k entries of the corresponding coefficient vector are 0. We use these known values to construct a new linear system

$$0 = \tilde{A} s_0 + \tilde{b}.$$

The attacker observes output until this linear system is overdetermined, and then solves it. A solution giving s_0 is guaranteed to exist. Although the solution is not always unique, in practice this method appears to recover a solution close to the initial state after $\lceil (n-k)/k \rceil$ outputs.

This attack is efficient. With $n = 1024$ and $k = 341$, recovering the FF-MS-PRG state from 9 outputs took 7 minutes implemented in Sage; the unoptimized construction of the linear system was the bottleneck.

5.4.2 The Bad-e (Be) attack.

In this section, we describe choices for the exponent e that lead to efficient state-recovery attacks for the Micali-Schnorr generator. The particular choices of e we make are unusual, but allowed by the ISO standard, and are efficient to exploit with the output sizes recommended by ISO.

As observed in Section 5.2, a straightforward application of multivariate Coppersmith's method for a state recovery attack against MS PRG is ruled out by the parameters specified by ISO. We can circumvent these restrictions by choosing a large e such that $e^{-1} \bmod \varphi(N)$ is small.

Flexible choice of e. ISO specifies that "The implementation should allow the application to request any *odd* integer e in the range $1 < e < 2^{\lg(N)-1} - 2 \cdot 2^{\lg N/2}$."

Our attack instantiates the public exponent e with a value other than the default exponent $e = 3$. Using larger e results in a larger output length k under the recommended parameters. Interestingly, while MS DRBG can be instantiated with almost any non-default e, there are more requirements on the public modulus: N may either be one of the default moduli or randomly generated.

$e = d^{-1}$ for small d is insecure.

Theorem 9. *Instantiating Micali-Schnorr with RSA using exponent $e = d^{-1} \bmod \varphi(N)$ for d small allows an efficient state recovery attack from a single output when the state has length $r < n/(\binom{d}{2} + 1)$, in time polynomial in d and $\lg N$.*

Proof. One output b_1 yields a degree-d polynomial relating states s_0 and s_1.

$$\begin{aligned} s_0^e &= (2^k s_1 + b_1) \bmod N \\ s_0 &= (2^k s_1 + b_1)^d \bmod N \\ s_0 - (2^k s_1 + b_1)^d &= 0 \bmod N \end{aligned}$$

A straightforward application of multivariate Coppersmith results in a lattice of dimension $d+2$ and determinant $R^{\binom{d}{2}+1} N^{d+1}$ for R the bound on the size of the state. Applying Condition 1 and omitting the approximation factor if we expect d to be a small constant, we expect to succeed when $r(\binom{d}{2} + 1) < n$. □

We can verify that this attack is allowed by the ISO parameters. When e is large, which is what we expect for $d^{-1} \bmod \varphi(N)$ for d small, the ISO parameters set $r = 2\gamma$ where γ is the security parameter. Thus we expect this attack to work when $\gamma(d^2 - d + 2) < n$. To be concrete, for the ISO security parameters (listed in Section 2), this inequality is satisfied for $d = 3$ for all parameter sizes, for $d = 5$ at $\gamma = 128$ and above, and $d = 7$ for $\gamma = 256$.

This exponent can be efficiently computed from knowledge of the factorization of N, and we expect it to be large since $\varphi(N) \mid de - 1$. This choice of e is arguably not a plausibly deniable "backdoor" since in practice e is almost always chosen to be small. In addition, it is efficiently discoverable for any small d via the Boneh and Durfee attack on small private RSA exponents [BD99]. However, in cryptographic protocols in which parameters are negotiated by machines making basic validity checks rather than actively looking for suspicious parameters, even such a discoverable backdoor could easily go undiscovered.

$e = e_0 e_1^{-1}$ is insecure. A full generalization of this attack would construct exponents of the form $e = e_0 e_1^{-1}$ for small multipliers e_0. This parameter generation is efficient when the factorization of N is known. This enables an efficiently exploitable state recovery attack using Coppersmith's method for parameters at ISO security levels $\gamma = 112$ through 256. However, this special-form exponent backdoor remains detectable, because it is possible for an external observer who suspects the form of the backdoor to brute force search over pairs of small e_0, e_1 to recover $\varphi(N)$ and thus factor N. Full details of these algorithms are given in Appendix A.4.

6 Impact on cryptographic protocols

Deployed cryptographic systems typically use random numbers as input to a cryptographic protocol. The precise interaction between protocol, implementation and a subverted RNG can have a major impact on the exploitability of a system. We now briefly consider how our attacks on MS and RSA PRG may affect common protocols.

Case study: Using MS DRBG state recovery to subvert IPsec. IPsec [DGT98] is an encryption protocol often used for VPNs. We focus on the key agreement protocol, typically IKE. During the period of standardization (approximately 2004-2007), the current version of the protocol was IKEv1 [HC98].

As noted in previous analyses [CNE+14, CMG+16, CGH18] many IKE implementations use a single PRG to generate both unencrypted nonces, encryption padding and ephemeral secret keys for Diffie-Hellman key agreement. An attacker wishing to passively exploit the "Bad-e" MS DRBG state recovery attack we outline in Section 5.4 would observe protocol handshakes from such an implementation, use the nonces to carry out a state recovery attack, and then iterate the state forward to recover the secret Diffie-Hellman exponent, recover the shared secret, and derive the symmetric session keys to decrypt the session data.

For the simplest attack described in Theorem 9, state recovery is feasible for all security parameters with exponent $e = 3^{-1} \bmod \varphi(N)$ and requires observing at least $3n/4$ bits of output. For $n = 1024$ this is 96 bytes and for $n = 2048$ it is 192 bytes, both within the 256-byte upper limit on a variable-length nonce.

Extracting generator output from public nonces. The most likely source for public output is the random nonces in each key agreement: these range from 8–256 bytes in IKE. Thus in IKE a single nonce is conceptually sufficient to recover a single generator output block using a 1024-bit or 2048-bit modulus.

Extracting generator output from RSA padding. Some configurations of the IKEv1 protocol employ RSA-PKCS#1v1.5 encryption to authenticate endpoints. In this configuration, one party encrypts a nonce to the other party's encryption key. Assuming an attacker can interact with the server once, it may therefore obtain raw PRG output in the padding of the RSA ciphertext.

With an RSA public key of length $\bar{n}$ bytes and a nonce of length m bytes, each ciphertext contains $\bar{n} - 3 - m$ bytes of *non-zero* RSA padding bytes, in addition to the m-byte nonce.[9] Assuming a 32-byte nonce, this provides 93 bytes of padding (or 125 bytes for padding and nonce combined) for a 1024-bit RSA encryption key and 221 bytes (or 253 bytes for both) for a 2048-bit encryption key. The 93-byte padding is less than the 96 output bytes required for a 1024-bit key, but the remaining bytes could conceivably be recovered via brute force.

Case Study: TLS. SSL/TLS [FKK11, AD99] are the most common secure communications protocols used on the Internet. SSL and TLS each combine the use of long-term keys or secrets, as well as a key agreement protocol and symmetric encryption scheme for transmission of secure data into a single protocol. Common versions between 2004–2007 included SSL version 3 [FKK11] and TLS 1.0-1.2 [AD99, DR06, RD08].

The random portion of an SSL/TLS nonce is 28 bytes long.[10] For SSL/TLS or IKE implementations with smaller nonces, an attacker would need to obtain several nonces over multiple key exchange interactions (approximately 4 at the 28 bytes length) in order to recover sufficient state to obtain one 108-byte MS DRBG output at the 1024-bit security level. Even this approach poses a challenge: for our basic attacks, the recovered output bytes must be *consecutive*. In a naive implementation of either protocol, the generation of nonces may be interspersed with other uses

[9] The PKCS#1v1.5 standard requires that all padding bytes be non-zero, since the 0 byte is used as a delimiter. Recovering the raw byte stream would thus require some additional steps depending on how this string is generated.

[10] Each nonce comprises 28 bytes of random data concatenated with a 4-byte timestamp.

of the PRG: as a result, only fragments of each output block would be available. There are two potential engineering solutions that could mitigate this result:

1. During the standardization period, the NSA proposed and co-authored numerous IETF draft extensions to SSL/TLS [RS09, RS06, HS09, Hof10, Hof12] that cause servers and clients to output much longer nonces on request. The motivation for these extensions is controversial [Pta15], but at least one extension was ultimately deployed in the BSAFE commercial cryptography library [Ben17]. The occasional use of such extensions by any client would provide eavesdroppers with an arbitrary amount of generator output that could be used to recover secret keys until the generator was reseeded.

2. Some commercial implementations of IKE *pre-generate* nonces in advance of a handshake, storing the results in a queue for later use [CMG+16]. Such implementations have been discovered in devices implementing the Dual EC DRBG generator, a design choice that maximizes the practical impact of a subversion attack. A similar implementation decision could allow the exfiltration of multiple consecutive bytes of generator output over several handshakes.

There are also some algorithmic exploitation possibilities:

1. Multivariate Coppersmith methods can be used to solve for multiple nonconsecutive chunks of output; the exact bounds would depend on the details of the implementation.

2. Our SUS backdoor for RSA PRG exploits sequences of non-consecutive blocks of output, albeit selected to satisfy the linear backdoor recurrence embedded in the modulus. A more moderate improvement in the bounds might allow a recovery attack of this form.

7 Conclusion

In this paper, we study the question of whether an adversary who controls the generation of the parameters used for the Micali-Schnorr PRG can break the security of the algorithm. To that end, we identify vulnerable parameters permitted by the ISO standard for Micali-Schnorr, and develop a novel backdoor algorithm for the closely-related RSA PRG that permits efficient state recovery attacks beyond previously known bounds. However, we encounter barriers in adapting our backdoor technique to MS PRG for realistic parameters, and thus the main question we set out to solve remains open.

A solution to this problem may involve the development of new ideas in the cryptanalysis of RSA. For example, the small characteristic finite field case has exploitable structure. Taking advantage of this structure leads to improvements in algorithms like the function field sieve for discrete logarithms over small-characteristic finite fields. An analogous improvement for the integers that allows simplifications of the recurrences might open doors (or be related to existing advances) in the study of factorization or RSA cryptanalysis algorithms.

Acknowledgments. We thank Emmanuel Thomé and Antonio Sanso as well as numerous attendees of CHES 2016 for enjoyable conversations about this problem. This work was supported by NSF under awards CNS-1653110, CNS-1801479, DMS-1913210, and CNS-2048563, and by DARPA under Contract No. HR001120C0084. Any opinions, findings and conclusions or recommendations

expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government or DARPA.

References

[ACGS84] Werner Alexi, Benny Chor, Oded Goldreich, and Claus-Peter Schnorr. RSA/Rabin bits are $1/2 + 1/\text{poly}(\log N)$ secure. In *25th FOCS*, pages 449–457. IEEE Computer Society Press, October 1984.

[AD99] Christopher Allen and Tim Dierks. The TLS Protocol Version 1.0. RFC 2246, January 1999.

[And93] Ross J. Anderson. Practical RSA trapdoor. *Electronics Letters*, 29:995–995, 1993.

[BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, 1986.

[BD99] Dan Boneh and Glenn Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 1–11. Springer, Heidelberg, May 1999.

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, Heidelberg, April 2008.

[Ben17] David Benjamin. Additional TLS 1.3 results from Chrome. `https://mailarchive.ietf.org/arch/msg/tls/i9blmvG2BEPf1s1OJkenHknRw9c/`, December 2017.

[BL17] Sebastian Berndt and Maciej Liskiewicz. Algorithm substitution attacks from a steganographic perspective. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1649–1660. ACM Press, October / November 2017.

[BLN16] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. *Dual EC: A Standardized Back Door*, pages 256–281. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd FOCS*, pages 112–117. IEEE Computer Society Press, November 1982.

[BPR14] 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. Springer, Heidelberg, August 2014.

[Ces22] Marco Cesati. A new idea for RSA backdoors. `https://arxiv.org/abs/2201.13153`, 2022.

[CGH18] Shaanan N. Cohney, Matthew D. Green, and Nadia Heninger. Practical state recovery attacks against legacy RNG implementations. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 265–280. ACM Press, October 2018.

[CMG+16] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. A systematic analysis of the juniper dual EC incident. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 468–479. ACM Press, October 2016.

[CNE+14] 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, *USENIX Security 2014*, pages 319–335. USENIX Association, August 2014.

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, September 1997.

[Cop01] Don Coppersmith. Finding small solutions to small degree polynomials. In Joseph H. Silverman, editor, *Cryptography and Lattices*, pages 20–31, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[CRT+19] Sherman S. M. Chow, Alexander Russell, Qiang Tang, Moti Yung, Yongjun Zhao, and Hong-Sheng Zhou. Let a non-barking watchdog bite: Cliptographic signatures with an offline watchdog. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 221–251. Springer, Heidelberg, April 2019.

[CS03] Claude Crépeau and Alain Slakmon. Simple backdoors for RSA key generation. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 403–416. Springer, Heidelberg, April 2003.

[DGG+15] 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. Springer, Heidelberg, April 2015.

[DGT98] Naganand Doraswamy, K. Robert Glenn, and Rodney L. Thayer. IP Security Document Roadmap. RFC 2411, November 1998.

[DPR+13] 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 Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 647–658. ACM Press, November 2013.

[DPSW16] Jean Paul Degabriele, Kenneth G. Paterson, Jacob C. N. Schuldt, and Joanne Woodage. Backdoors in pseudorandom number generators: Possibility and impossibility results. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 403–432. Springer, Heidelberg, August 2016.

[DR06] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, April 2006.

[drb] DRBG recommmendations from the x9.82 editing group. Document online, `https://github.com/matthewdgreen/nistfoia/blob/master/6.4.2014%20production/055%20-%20DRBG%20Recomm%20from%20X9.82%20Editing%20Group.pdf`.

[Eng20] Lynn Engelberts. Analysis of the Micali-Schnorr PRNG with known factorisation of the modulus. Master's thesis, University of Oxford, 2020.

[FKK11] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101, August 2011.

[FM18] Marc Fischlin and Sogol Mazaheri. Self-guarding cryptographic protocols against algorithm substitution attacks. In Steve Chong and Stephanie Delaune, editors, *CSF 2018 Computer Security Foundations Symposium*, pages 76–90. IEEE Computer Society Press, 2018.

[FS97] Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for RSA and rabin bits. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 267–279. Springer, Heidelberg, May 1997.

[FS00] Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, March 2000.

[FVZ13] Pierre-Alain Fouque, Damien Vergnaud, and Jean-Christophe Zapalowicz. Time/memory/data tradeoffs for variants of the RSA problem. In Ding-Zhu Du and Guochuan Zhang, editors, *Computing and Combinatorics*, pages 651–662, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[FZ14] Pierre-Alain Fouque and Jean-Christophe Zapalowicz. Statistical properties of short RSA distribution and their cryptographic applications. In Zhipeng Cai, Alex Zelikovsky, and Anu Bourgeois, editors, *Computing and Combinatorics*, pages 525–536, Cham, 2014. Springer International Publishing.

[Gre13] Matthew Green. A few more notes on NSA random number generators. https://web.archive.org/web/20230109062504/https://blog.cryptographyengineering.com/2013/12/28 few-more-notes-on-nsa-random-number/, December 2013.

[har] DRBGs based on hard problems. Document online, `https://github.com/matthewdgreen/nistfoia/blob/master/6.4.2014%20production/039%20-%20DRBGs%20Based%20on%20Hard%20Problems.pdf`.

[HC98] Dan Harkins and Dave Carrel. The Internet Key Exchange (IKE). IETF RFC 2409 (Proposed Standard), 1998.

[HM09] Mathias Herrmann and Alexander May. Attacking power generators using unravelled linearization: When do we output too much? In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 487–504. Springer, Heidelberg, December 2009.

[Hof10] Paul E. Hoffman. Additional Random Extension to TLS. Internet-Draft draft-hoffman-tls-additional-random-ext-01, Internet Engineering Task Force, February 2010. Work in Progress.

[Hof12] Paul E. Hoffman. Additional Master Secret Inputs for TLS. RFC 6358, January 2012.

[HS09] Paul E. Hoffman and Jerome Solinas. Additional PRF Inputs for TLS. Internet-Draft draft-solinas-tls-additional-prf-input-01, Internet Engineering Task Force, October 2009. Work in Progress.

[Int11] International Organization for Standardization. ISO/IEC 18031:2011 information technology—security techniques—random bit generation. Online documents., 2011. `https://www.iso.org/standard/54945.html`.

[Joh04] Don B. Johnson. X9.82 part 3: Number theoretic DRBGs. Presented at the NIST RNG Workshop, July 2004. Slides online: `https://csrc.nist.gov/CSRC/media/Events/Random-Number-Generation-Workshop-2004/documents/NumberTheoreticDRBG.pdf`.

[Joy08] Marc Joye. Rsa moduli with a predetermined portion: Techniques and applications. In Liqun Chen, Yi Mu, and Willy Susilo, editors, *Information Security Practice and Experience*, pages 116–130, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Jut98] Charanjit S. Jutla. On finding small solutions of modular multivariate polynomial equations. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 158–170. Springer, Heidelberg, May / June 1998.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982.

[MS90] Silvio Micali and Claus-Peter Schnorr. Efficient, perfect random number generators. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 173–198. Springer, Heidelberg, August 1990.

[MS91] Silvio Micali and Claus-Peter Schnorr. Efficient, perfect polynomial random number generators. *Journal of Cryptology*, 3(3):157–172, January 1991.

[Nat15] National Institute of Standards and Technology. Results of a recent FOIA for NIST documents related to the design of Dual EC DRBG. Online documents., 2015. `https://github.com/matthewdgreen/nistfoia/`.

[NS06] Phong Q. Nguyen and Damien Stehlé. Lll on the average. In Florian Hess, Sebastian Pauli, and Michael Pohst, editors, *Algorithmic Number Theory*, pages 238–256, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Pat12] Constantinos Patsakis. Number theoretic SETUPs for RSA like factoring based algorithms. *J. Inf. Hiding Multim. Signal Process.*, 3(2):191–204, 2012.

[PLS13] Nicole Perlroth, Jeff Larson, and Scott Shane. N.S.A. able to foil basic safeguards of privacy on web. *New York Times*, September 2013.

[Pta15] Thomas Ptacek. Is Extended Random A Malicious NSA Plot?, August 2015.

[PW20] Thomas Peyrin and Haoyang Wang. The MALICIOUS framework: Embedding backdoors into tweakable block ciphers. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 249–278. Springer, Heidelberg, August 2020.

[RD08] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[RH23] Keegan Ryan and Nadia Heninger. Fast practical lattice reduction through iterated compression. Cryptology ePrint Archive, Report 2023/237, 2023. `https://eprint.iacr.org/2023/237`.

[RS06] Eric Rescorla and Margaret Salter. Opaque PRF Inputs for TLS. Internet-Draft draft-rescorla-tls-opaque-prf-input-00, Internet Engineering Task Force, December 2006. Work in Progress.

[RS09] Eric Rescorla and Margaret Salter. Extended Random Values for TLS. Internet-Draft draft-rescorla-tls-extended-random-02, Internet Engineering Task Force, March 2009. Work in Progress.

[RTYZ16] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 34–64. Springer, Heidelberg, December 2016.

[San17] Antonio Sanso. How to try to predict the output of micali-schnorr generator (MS-DRBG) knowing the factorization, 2017. `http://blog.intothesymmetry.com/2017/12/how-to-try-to-predict-output-of-micali.html`.

[SF07] Dan Shumow and Niels Ferguson. On the possibility of a back door in the NIST SP800-90 Dual Ec Prng. Presented at the Crypto 2007 rump session, August 2007. Slides online: `http://rump2007.cr.yp.to/15-shumow.pdf`.

[sha15] SHA-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 2015.

[SIG13] Excerpt from 2013 intelligence budget request: SIGINT ENABLING. `https://archive.nytimes.com/www.nytimes.com/interactive/2013/09/05/us/documents-reveal-nsa-campaign-against-encryption.html` Media leak, 2013.

[SPW06] Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. On the provable security of an efficient RSA-based pseudorandom generator. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 194–209. Springer, Heidelberg, December 2006.

[WKM16] Stefan Wüller, Marián Kühnel, and Ulrike Meyer. Information hiding in the RSA modulus. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, pages 159–167, 2016.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, November 1982.

[YY96] Adam Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 89–103. Springer, Heidelberg, August 1996.

[YY97] Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 62–74. Springer, Heidelberg, May 1997.

[YY06] Adam Young and Moti Yung. A space efficient backdoor in RSA and its applications. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 128–143. Springer, Heidelberg, August 2006.

A Appendix

A.1 Proof of Theorem 3

In the initial game, Game 0, the adversary calls the **Init** oracle exactly once at the start of the game to obtain its initial input string B. It can query the **Func** oracle with an input X and a sign $\text{sgn} \in \{+,-\}$ to get output $F^{\text{sgn1}}(X)$. For convenience, we allow **Func** to sample its outputs lazily with a table T, as long as it maintains the permutation property.

It is clear that

$$|\Pr[\mathcal{A}[\mathbf{Func}_0](\mathbf{Init}_0()) \Rightarrow 1] - \Pr[\mathcal{A}[F, F^{-1}](B) \Rightarrow 1 | b \leftarrow \$\{0,1\}^{hk}]|$$

$\underline{\mathbf{Init}_0()}$

1. $b_1 \ldots b_h \leftarrow \$\{0,1\}^k$
2. $T \leftarrow []$
3. $S^+, S^- \leftarrow \{\}$
4. return $b_1 || \ldots || b_h$

$\underline{\mathbf{Func}_0(X, \text{sgn})}$

1. if $T[\text{sgn},X] \neq \perp$ then return $T[\text{sgn}, X]$
2. out $\leftarrow T[\text{sgn},X] \leftarrow \$[1, N] \setminus S^{\text{sgn}}$
3. $T[-\text{sgn},T[\text{sgn},X]] \leftarrow X$
4. $S^{\text{sgn}} \leftarrow S^{\text{sgn}} \cup \{T[\text{sgn}, X]\}$
5. $S^{-\text{sgn}} \leftarrow S^{-\text{sgn}} \cup \{X\}$
6. return out

In Game 1, the **Init** oracle chooses b_i by sampling uniformly random z_i and setting $b_i = z_i \mod 2^k$ to its low-order bits. The Micali-Schnorr paper states that this distribution is statistically indistinguishable from b_i with negligible bound ε, so

$$|\Pr[\mathcal{A}[\mathbf{Func}_1](\mathbf{Init}_1()) \Rightarrow 1] - \Pr[\mathcal{A}[\mathbf{Func}_0](\mathbf{Init}_0()) \Rightarrow 1]| \leq \varepsilon.$$

We also pick a random seed s from $[1, N2^{-k}]$, which will be used in later games. Additionally, the $\mathbf{Func}_1$ oracle sets a bad flag if any queries or responses collide with s, any z_i or their high-order bits. This is an internal, administrative change and is undetectable.

Init$_1()$

1. $z_0, \ldots, z_h \leftarrow \$[1, N]$
2. $s \leftarrow \$[1, N2^{-k}]$
3. for i from 1 to h
4. $\quad b_i \leftarrow z_i \mod 2^k$
5. $T \leftarrow []$
6. $S^+, S^- \leftarrow \{\}$
7. return $b_1 || \ldots || b_h$

Func$_1(X,$ sgn$)$

1. if $T[\text{sgn},X] \neq \perp$ then return $T[\text{sgn},X]$
2. out $\leftarrow T[\text{sgn},X] \leftarrow \$[1, N] \setminus S^{\text{sgn}}$
3. for i from 0 to h
4. $\quad$ if out $= z_i$ or out $= \lfloor 2^{-k} z_i \rfloor + 1$
5. $\quad\quad$ bad $\leftarrow$ true
6. $\quad$ if $X = z_i$ or $X = s$
7. $\quad\quad$ bad $\leftarrow$ true
8. $\quad$ if $X = \lfloor 2^{-k} z_i \rfloor + 1$
9. $\quad\quad$ bad $\leftarrow$ true
10. $T[-\text{sgn},T[\text{sgn},X]] \leftarrow X$
11. $S^{\text{sgn}} \leftarrow S^{\text{sgn}} \cup \{T[\text{sgn}, X]\}$
12. $S^{-\text{sgn}} \leftarrow S^{-\text{sgn}} \cup \{X\}$
13. return out

In Game 2, the **Func**$_2$ oracle programs the function F when the bad flag is set through the tables T and IT. By the fundamental lemma of game playing, it holds that:

$$|\Pr[\mathcal{A}[\mathbf{Func}_2](\mathbf{Init}_2()) \Rightarrow 1] - \Pr[\mathcal{A}[\mathbf{Func}_1](\mathbf{Init}_1()) \Rightarrow 1]| \leq \Pr[\text{ bad is set in Game 1}].$$

We give an upper bound for this probability.

For any output string $b_1 \ldots, b_h$, there are at least $\lfloor N2^{-k} \rfloor$ possible values for each z_i, and all of these values are equally likely and independent of the output of the **Func**$_1$ oracle. The probability that any query X to **Func**$_1$ equals any z_i can therefore be upper bounded by $\frac{h}{N2^{-k}}$ using a union bound.

For each b_i, there are at least $\lfloor N2^{-k} \rfloor - 1$ possible values for $\lfloor 2^{-k} z_i \rfloor$. Each of these occurs with probability at most $\frac{2^k+1}{N}$, and they are likewise independent of **Func**'s outputs. Therefore the probability that any query to **Func**$_1$ equals $\lfloor 2^{-k} z_i \rfloor + 1$ for some z_i is bounded above by $\frac{h+h2^k}{N}$.

The bad flag can also be set if the **Func**$_1$ oracle's random choices collide with z_i or $\lfloor 2^{-k} z_i \rfloor + 1$. Each value of *out* is sampled independently of the values of z_i, and it is chosen uniformly from a set of size at least $N - q$. In any query, there are $2h$ values that will trigger a collision. Then probability that such a collision occurs over all queries is at most $\frac{2hq}{(N-q)}$. Therefore, the total probability that the bad flag is set in Game 1 is at most $\frac{2hq}{N2^{-k}} + \frac{hq}{N} + \frac{2hq}{N-q}$, and

$$|\Pr[\mathcal{A}[\mathbf{Func}_2](\mathbf{Init}_2()) \Rightarrow 1] - \Pr[\mathcal{A}[\mathbf{Func}_1](\mathbf{Init}_1()) \Rightarrow 1]| \leq \frac{2hq}{N2^{-k}} + \frac{hq}{N} + \frac{2hq}{N-q}.$$

$\underline{\textbf{Func}_2(X,\ \text{sgn})}$

1. if $T[\text{sgn},X] \neq \perp$ then return $T[\text{sgn},X]$
2. out $\leftarrow T[\text{sgn},X] \leftarrow \$[1,N] \setminus S^{\text{sgn}}$
3. for i from 0 to h
4. if out $= z_i$ or out $= \lfloor 2^{-k} z_i \rfloor + 1$
5. bad $\leftarrow$ true
6. if $sgn = +$ and out $= z_i$
7. $out \leftarrow T[+,X] \leftarrow \$[1,N] \setminus \{S^+ \cup \{z_i\}_{i=1}^h\}$
8. if $sgn = -$ and out $= \lfloor 2^{-k} z_i \rfloor + 1$
9. $out \leftarrow T[-,X] \leftarrow \$[1,N] \setminus \{S^- \cup \{\lfloor 2^{-k} z_i \rfloor + 1\}_{i=1}^h\}$
10. if $X = z_i$ or $X = s$
11. bad $\leftarrow$ true
12. if $sgn = -$ and $i > 0$ and $X = z_i$
13. out $\leftarrow T[-,X] \leftarrow \lfloor 2^{-k} z_{i-1} \rfloor + 1$
14. else if $sgn = +$ and $X = s$
15. out $\leftarrow T[+,X] \leftarrow z_1$
16. if $X = \lfloor 2^{-k} z_i \rfloor + 1$
17. bad $\leftarrow$ true
18. if $sgn = +$ and $i < h$
19. out $\leftarrow T[+,X] \leftarrow z_{i+1}$
20. $T[-\text{sgn},T[\text{sgn},X]] \leftarrow X$
21. $S^{\text{sgn}} \leftarrow S^{\text{sgn}} \cup \{T[\text{sgn},X]\}$
22. $S^{-\text{sgn}} \leftarrow S^{-\text{sgn}} \cup \{X\}$
23. return out

In Game 3, the $\textbf{Init}_3$ oracle programs F so that $z_i = F(\lfloor 2^{-k} z_{i-1} \rfloor + 1)$. This is just bookkeeping: certain entries of tables T and T^{-1} are set earlier than they would be set by the $\textbf{Func}_3$ oracle, and some entries of T are set that would not be set by **Func**. Since the adversary cannot tell which entries of T have been initialized (except for its trivial knowledge of which entries have been initialized by queries to $\textbf{Func}_3$), these changes are undetectable. Therefore

$$\Pr[\mathcal{A}[\textbf{Func}_3](\textbf{Init}_3() \Rightarrow 1] = \Pr[\mathcal{A}[\textbf{Func}_2](\textbf{Init}_2() \Rightarrow 1].$$

$\underline{\mathbf{Init}_3()}$

1. $z_0, \ldots, z_h \leftarrow \$[1, N]$
2. $s \leftarrow \$[1, N2^{-k}]$
3. $T \leftarrow []$
4. $S^+, S^- \leftarrow \{z_1\}, \{s\}$
5. $T[+, s] \leftarrow z_1$
6. $T[-, z_1] \leftarrow s$
7. for i from 1 to h
8. $\quad b_i \leftarrow z_i \mod 2^k$
9. $\quad$ if $i > 0$ then $T[-, z_i] \leftarrow \lfloor 2^{-k} z_{i-1} \rfloor + 1$
10. $\quad T[+, T[-, z_i]] \leftarrow z_i$
11. $\quad S^+ \leftarrow S^+ \cup \{z_i\}$
12. $\quad S^- \leftarrow S^- \cup \{\lfloor 2^{-k} z_i \rfloor + 1\}$
13. return $b_1 || \ldots || b_h$

Finally, in Game 4, we can remove all unreachable code from the **Func** oracle. All the programming in lines 3-17 is now unreachable, since queries on any of the relevant points have been preprogrammed by **Init**. This pruning does not alter the behavior of any oracle.

$$\Pr[\mathcal{A}[\mathbf{Func}_4](\mathbf{Init}_4()) \Rightarrow 1] = \Pr[\mathcal{A}[\mathbf{Func}_3](\mathbf{Init}_3()) \Rightarrow 1]$$

Init$_4$()

1. $z_0, \ldots, z_h \leftarrow \$[1, N]$
2. $s \leftarrow \$[1, N2^{-k}]$
3. $T \leftarrow []$
4. $S^+, S^- \leftarrow \{z_1\}, \{s\}$
5. $T[+, s] \leftarrow z_1$
6. $T[-, z_1] \leftarrow s$
7. for i from 1 to h
8. $\quad b_i \leftarrow z_i \mod 2^k$
9. $\quad$ if $i > 1$ then $T[-, z_i] \leftarrow \lfloor 2^{-k} z_{i-1} \rfloor + 1$
10. $\quad T[+, T[-, z_i]] \leftarrow z_i$
11. $\quad S^+ \leftarrow S^+ \cup \{z_i\}$
12. $\quad S^- \leftarrow S^- \cup \{\lfloor 2^{-k} z_i \rfloor + 1\}$
13. return $b_1 || \ldots || b_h$

Func$_4(X,$ sgn$)$

1. if $T[\text{sgn},X] \neq \perp$ then return $T[sgn,X]$
2. out $\leftarrow T[\text{sgn},X] \leftarrow \$[1, N] \setminus S^{\text{sgn}}$
3. $T[-\text{sgn},T[\text{sgn},X]] \leftarrow X$
4. $S^{\text{sgn}} \leftarrow S^{\text{sgn}} \cup \{T[\text{sgn}, X]\}$
5. $S^{-\text{sgn}} \leftarrow S^{-\text{sgn}} \cup \{X\}$
6. return out

We now claim that the **Init** oracle of Game 4 honestly simulates the Micali-Schnorr PRG instantiated with the lazily-sampled random permutation F defined by table T, except in the case where two of $z_0, \ldots, z_h$ collide. We implicitly define internal states $s_{i+1} = \lfloor 2^{-k} z_i \rfloor + 1$ for i from 1 to h, and the initial seed is s.

By the birthday and union bounds, this happens with probability at most $\frac{(h+1)^2}{2N}$. Since the **Func** oracle faithfully computes F and F^{-1} on its inputs, we have that

$$|\Pr[\mathcal{A}[F](PRG[F](z_0) \Rightarrow 1) | z_0 \leftarrow \$[1, N2^{-k}] - \Pr[\mathcal{A}[\mathbf{Func}_4](\mathbf{Init}_4) \Rightarrow 1]| \leq \frac{(h+1)^2}{2N}.$$

Collecting bounds from earlier gamehops gives us that $\mathrm{Adv}_F^{prg}(\mathcal{A}) =$

$$
\begin{aligned}
& |\Pr[\mathcal{A}[F](PRG[F,F^{-1}]() \Rightarrow 1)] - Pr[\mathcal{A}[F,F^{-1}](B) \Rightarrow 1 | B \leftarrow \$\{0,1\}^{hk}] \\
\leq\ & |\Pr[\mathcal{A}[F](PRG[F,F^{-1}]() \Rightarrow 1)] - \Pr[\mathcal{A}[\mathbf{Func}_4](\mathbf{Init}_4()) \Rightarrow 1]| \\
& + |\Pr[\mathcal{A}[\mathbf{Func}_4](\mathbf{Init}_4()) \Rightarrow 1] - Pr[\mathcal{A}[F,F^{-}1](B) \Rightarrow 1 | b \leftarrow \$\{0,1\}^{hk}]| \\
\leq\ & \frac{(h+1)^2}{2N} + |\Pr[\mathcal{A}[\mathbf{Func}_4](\mathbf{Init}_4()) \Rightarrow 1] - Pr[\mathcal{A}[F](B) \Rightarrow 1 | b \leftarrow \$\{0,1\}^{hk}]| \\
\leq\ & \frac{(h+1)^2}{2N} + |\Pr[\mathcal{A}[\mathbf{Func}_3](\mathbf{Init}_3()) \Rightarrow 1] - Pr[\mathcal{A}[F,F^{-1}](B) \Rightarrow 1 | b \leftarrow \$\{0,1\}^{hk}]| \\
\leq\ & \frac{(h+1)^2}{2N} + |\Pr[\mathcal{A}[\mathbf{Func}_2](\mathbf{Init}_2()) \Rightarrow 1] - Pr[\mathcal{A}[F,F^{-1}](B) \Rightarrow 1 | b \leftarrow \$\{0,1\}^{hk}]| \\
\leq\ & \frac{(h+1)^2}{2N} + |\Pr[\mathcal{A}[\mathbf{Func}_2](\mathbf{Init}_2()) \Rightarrow 1] - \Pr[\mathcal{A}[\mathbf{Func}_1](\mathbf{Init}_1()) \Rightarrow 1]| \\
& + |\Pr[\mathcal{A}[\mathbf{Func}_1](\mathbf{Init}_1()) \Rightarrow 1] - Pr[\mathcal{A}[F,F^{-1}](B) \Rightarrow 1 | b \leftarrow \$\{0,1\}^{hk}]| \\
\leq\ & \frac{(h+1)^2}{2N} + \frac{2hq}{N2^{-k}} + \frac{hq}{N} + \frac{2hq}{N-q} \\
& + |\Pr[\mathcal{A}[\mathbf{Func}_1](\mathbf{Init}_1()) \Rightarrow 1] - Pr[\mathcal{A}[F,F^{-1}](B) \Rightarrow 1 | b \leftarrow \$\{0,1\}^{hk}]| \\
\leq\ & \frac{(h+1)^2}{2N} + \frac{2hq}{N2^{-k}} + \frac{hq}{N} + \frac{2hq}{N-q} + \varepsilon.
\end{aligned}
$$

For values of q and h that are polynomial in $n = \lg N$, and k within the ISO standardized range, this advantage is negligible.

A.2 Example parameters for eSP attack

The following 2048-bit modulus $N = pq$ was generated such that $5^{504} \equiv 1 \pmod{p-1}$ and $5^{504} \equiv 187 \pmod{q-1}$, for use in the eSP attack with exponent $e = 5, c_q = 187$, and cycle length $\ell = 504$:

```
N =
0xfa4351292767d540d8dd2ef15b39b02e29b56ad2125add6964203eb0029ba7aa8d44c8e5
  65df7818f6eb959c37f83349ec4b1514ed42741b8e772028db779f5e362234b782b9064c
  ef07dead66bced100eb71cc8bf7f1325959eb304e90ecf6e53c4eff9708ebd3c7641ea2b
  aed50fbc679b4f0e06ad60b8e70c7836a5f99e1571be6194afe04c81a6eef8281bb43b99
  f17ee715e5f1a95e36aee8c4eb8757aa8ff108acdf29136cf543bb8831074b71fc63cc44
  4ca4d3d7811b6cdca33ec889dddc77828510ff0d10ac07b0f691ce2995a72137abf32816
  eed9cd322075e7450326ff2fba0b65c19c477d61407eccf48f5857b5ff95cf4c63702b9f
  09f9ca83

p =
0x9da439c726a6c69087a81e49644d5e2d28937de5ba08c475a615813025318e2ae05f3174
  8f685c68f02b6883f10d407012970b50b28da780a6b157b7c80011a3857a877d56b5a8d4
  fc0a96bf62b7ec86d735de29628e230665356f535df7664723659f8f88847baaf9526393
  a521

q =
0x1966956586a77b7e1fda36f722223a66b161f10027c53b2b39fde757e01fa6ef8b79e62e
```

```
0b423546ebc2473011c3a79b4379788fff85eb743bfdbfe1d679511042295e5c6a5cf634
b650bf83a997ecef1abef4467e02103de493f1f269bdb9088c6bc0f114971d05c983f499
a48551bdeb901d3bc3290fbbf49368e5d9495b3922bdc03a9ec38e49aee6f0cedc78a1e2
c5723
```

We used 504 for this example because it is highly composite and so $x^{504}-1$ has small polynomial factors, which makes $5^{504}-1$ much easier to factor. Example verification code for these parameters can be found in Appendix A.5.

A straightforward algorithm for generating such a modulus would be similar to Algorithm 4:

Algorithm 7: eSP parameter generation.

1 Choose an exponent e and cycle length ℓ.
2 Generate a prime p using Algorithm 4.
3 Choose small $c_q > 1$, such that finding roots of degree c_q polynomials mod q is feasible.
4 Take subsets of factors of $e^\ell - c_q$, until a subset is found where $q \triangleq 1 + \prod_i q_i$ is prime.
(This ensures $e^\ell \equiv c_q \pmod{q-1}$.)

Output: Public information $(N = pq, e)$,
secret backdoor information (p, q, ℓ, c_q).

However, the polynomial $x^\ell - c_q$ for $c_q > 1$ tends not to factor as well as $x^\ell - 1$. This means $e^\ell - c_q$ tends not to have as many small factors, so Algorithm 7 is much slower than generating two primes using Algorithm 4, especially if the output needs to be precisely a certain size.

To quickly make N exactly 2048 bits, we used the following alternative algorithm for Algorithm 7: Instead of first choosing c_q and then trying to find q of the right size, choose d such that $q \approx e^\ell/d$ would make pq the right size, then set q to the first prime less than e^ℓ/d. Then $c_q = e^\ell \bmod (q-1) \approx d(q - e^\ell/d)$, which is likely to be small for appropriately chosen parameters.

Constructing this exactly 2048-bit N with such a small c_q required a few hours of trial and error to find the appropriate parameters, but we believe it should be possible to improve this.

A.3 SUS Attack Parameter Selection

We give below a variety of parameters illustrating the tradeoffs in SUS parameters. The modulus sizes and output lengths are chosen according to the ISO standard for MS DRBG. The parameters were generated using the informal psuedocode in Algorithm 8.

15360-bit N, 2^{51} undiscoverability The MS DRBG uses this modulus size for the 256-bit security level.

$f(x) = x^{3500} + x^{3160} - x^{2580} + x^{2443} - x^{2043} - x^{1070}$.

Highest block of output needed is the 3500th.

We use $t = 3$ for Coppersmith multiplicity, giving lattice dimension 442.

We use exponent $e = 41$; outputs are $k = 14608$ bits, so that unknown part of each state is 752 bits — this is $2n/e$, rounded up to the next multiple of 8, as would be used in ISO MS DRBG.

Exploitation took 4 core-hours on an Intel Xeon E5-2699 v4 processor, using the `flatter` lattice reduction library[11] [RH23].

q can be any 901-bit prime; p is the following 14489-bit prime, such that $f(e) \equiv 0 \bmod \varphi(p)$:

[11] `https://github.com/keeganryan/flatter`

Algorithm 8: Strategy for generating SUS-backdoored parameters

1 For a given security level λ, look up the standard value of $\lg N$.
2 For undiscoverability, N/p must be too large for ECM to find — say, 300 bits — giving an upper bound on $\lg p$.
3 Under ISO MS DRBG parameters, the unknown states are always at least 2λ bits (and when e is small enough, as large as $(2 \lg N)/e$ bits). Pick a plausibly small e, which might need to be adjusted later. This determines the size R of the unknowns.
4 For a given number of nonzero terms m in the backdoor polynomial, and for this R and upper bound on $\lg p$, we can try various lattice constructions (by varying the multiplicity and polynomial shifts). Using Condition 1 we can check how large the modulus p must be for Multivariate Coppersmith to be expected to succeed. If this lower bound on $\lg p$ is less than the upper bound on $\lg p$ from earlier, and if the lattice dimension is small enough for LLL to be practical, then exploitation is possible for this m. Use the largest possible value of m for better undiscoverability, or use smaller m (which gives smaller lattice dimension) for more practical exploitation.
5 If practical exploitation is impossible even for $m = 4$ (the smallest nontrivial m), try again with a larger e.
6 Pick a degree bound d for the backdoor polynomial. The entropy of the backdoor polynomial is $\lg \binom{d}{k-1}$. Increasing d gives the backdoor slightly more entropy (better undiscoverability), but exploiting it requires PRG outputs as high as the dth block of output (less practical exploitation). Moreover, if e^d is more than a few hundred bits larger than $\lg p$, parameter generation may become harder.
7 Generate a random monic polynomial f of degree d with m nonzero coefficients, half of which are $+1$ and half of which are -1.
8 Find prime factors of $f(e)$ using ECM until the rough part Q is smaller than the upper bound on $\lg p$ (ideally, much smaller).
9 Try combinations of the factors found by ECM until $p = Q \times$ (other factors) $+ 1$ is a prime of the correct size.
10 Generate a random prime q so that $pq = N$ is the correct size.
Output: Public parameters (N, e); secret backdoor information (f, p, q).

74e296864a75a92b0bbe2be4e81c58a0e1c846cddfbe7b22d67fa97dc1ec19006aa74dd1eef5ddbf5e47c1b5f20fa69f0e66f6b6e94e998ce5b3e250d22aa5b0
87a8ed6f74137178f40c690c97dd087f44bda92223c79161582a9588bba4667a357a7273d4fe92e11a2157bb73d369442b6bc12c737ec9311442e3619562d62e
51f43f7c0298559fc347c1390e796c408c7f105ac32a21c8221d661d4334537c8a4c298611da287389190fb1a8cc087df9d2ad7d6e3cd8fc7ff82afd5e5bb163
ab6a075e069ab30568b0664e5640c9c34d93e285d4849dd223232a54faf53c501725d6b141188c8d7480b45497fee122ef7783f00d87cd04c37afd7be98b58cc
58573110d931be42a5027d746201f871a9a4beadd3497d51a291ee6df0cc94a233ba86fdcad7a94ad81f0b9f76d804fc15bb12bc93bdd587e312eb544bb290da
8beb77ab5d0e5d07b3f5a00f115200f13840c389d91685670e93e9c6754b4cec0c5b99a54b902b04becb7a3b639b0c1a370588a98d38124092c7a79cd4aeaa6e
06b782b602636249eab38ffbbf360e9fbf56b1294fac8f82d1e76d3173afc495c763e2ddb2b528740b4b9bff95166fe464a2baf7dc248625415e28c6c23c2603c
1033547824172de2716111594518dff62125f8cbf8320b60daa1a01f7fa9c1d659bc0eeb3078514c160e6eefa426eaa720e47e14554206e28dfbfb86fded6867
1a34d0d4c1cac36af664eafc3e59b240902aaed87ae4cb2f5d7d1844c62fa4859d55c4a74c3664c812f50c78635529f1c473275b4716d09ac5dbe01322d04ee5
ae9c7a54729b9baa30f84a5af0a5fc0ff01e28ab0f2ee2e47081c551c2a7ede326bb6aaabb927e1b687a60511f4a3aa2a6890c111f06986020c91dc871bbbf3e
7991775cdc52d4459a8584b8c538db61baa78e85e51281113854ba972f9dc6c4736b6a93fac78e3d6285063faaf559fb3bdce11e0529bdb4bb7aad46254a99d2
06d832ece82ac777ddec46dd4bea4320f58d9e9d60c38edef733f03d6d4823944b2eab82ce5d5bed7be5ec4ff8f32e22db44e4163622605fe898d1e5ff3faa64
cd5958b0612cc281df7e2ba2473c1f115b8e135f93d861704fdfeb7eb2489e47ed390b3c22468ba7656db0ff7fce1bd7a3c2469a0b5920eb6cc94f9e432ec579
f058fb16c682ab18720f9c237a59909bf6df7873a539139e5bdf489e59ee51f2ab0ec169d6dcceee32aee446a163b80854b86a9a05e7440d8de40c05c0f8e225
a672a1594e64974b6427814fcbd6fe83314a13f8aa4b28d884b1090d32917da93141731086f4cb0430e0e7960ad90463c85741e4dd1dd3744d494ec3824a4115
43b67ffbc04737042fd276a5712db5402cd386e9dab9303256f6c98c830f75adfabdb0c5b289ba85bf27289749d9ba22eff7b3045f8712ad99bf0afb217948ed
690eebea5aa089b53e502e151a7578ee0777faa4c9281ab1e959f05366444123c0eda329a99b6edfbabd67ed52bdb25b821fb1914f2de1e8cf26f04b21a8716d
fefc99737426103a48dc30f73142ce35bbd262b2b163776e948f534113acf1d85a675e2df7a7c57d80823085587099f947ee53d316c10128315b78706fb717f9
fbee12ea6ccaacbf1fbc3109a98a024375f0195393003657aabea5dac138c37ad92fd41f7ee440802ada7285f16082e0df0f3b9859faf9344b7129935725fa60
e17c141023679ca9a19022e43d5d790db814df33112fd3cb45d90e533826bc7321bc4f924bcde4e361e7b369cb7606674106f2a7a56c9d066e44bb7215e4a294
2159dad9c1635bfb1fb1ce072064ba3bb5cba0a3e11a3a7499836554055f09cff760b1fc394b3af8eb25079801fc5be37ab159b3d990b02baaa8ee8a7f9596b9
e2bc7e0eba4282323598817e246cd716ef632bb77b5bbd40bf386808a03d939353b8a80e4766a617d61004e723eec6ecd856f79acac1b4270a896fc1bf71d701
8f06e594ad226a8293426050af2dfc4e4d9b9d7c1a1e25dba48ab39ad5f3b2414e7e2e93c1c068b804da36fa5e0d669c8e7addb57670fdba352396ea60430026
d93ea0ec5761209e92192f5272f8ec17bfb6f14b9c2ff325bf15ae654f92d203f25b09834dea537bca75ee85c8cdea773593e4691948a09b0666df9609c1da38
a891524d683ac6811dba5641d507dee4304d3ad99eeddb4c6ef9b4e02e27893318d6dc4d2f7c10f9a9d61444054d877ce05056bb1d17f0734767032c0d754d8c
a6efb0b5f7410cb93980d677d58fea806df6d8a064d95c687ce99951e616c71762fdbbd93bbf2b8fbd9aa67aeefe3a417f28478d534f7fee12dd0c0520ebaf92
a45da4d54720c0792a1655f04ef713583447c3cabff548d008a6830612006e4d61c8300a452f9db7cac30396f37d91d1aa73671c070ca6edd33a0d904a6b3b35
e0463145423ff32a2bfe95e9f5395bae978f7dc63d5f68bee3f4c1b6752031cacd9806409136e513e056b5b2fd8d645ef3e3829a3c090c2020aab8e794dbce01
4913e2453950cc33dc50b3c4e0bc815

15360-bit N, 2^{50} undiscoverability, smaller outputs If we use $e = 39$ instead, so that the unknown part of each state is 788 bits, the multivariate Coppersmith attack no longer is expected to work with multiplicity 3, so we use multiplicity 4, giving a 1443-dimensional lattice.

$f(x) = x^{2850} + x^{2499} - x^{2025} - x^{1495} - x^{243} + x^{173}$

Highest block of output needed is the 2850th.

Exponent $e = 39$; outputs are 14572 bits (so that unknown part of each state is 788 bits).

We use multiplicity $t = 4$ for Coppersmith, giving lattice dimension 1443.

Exploitation took low hundreds of core-hours on an Intel Xeon E5-2699 v4 processor, again using `flatter`.

q can be any 1023-bit prime, p is the following 14337-bit prime such that $f(e) \equiv 0 \bmod \varphi(p)$:

1352e7566c191082295921c08f33ebe227d7e281753cb5325912c2e7ae997466c9d0379fd31f5690687d2150c2c4e3f90e7185a1bcbc3bb52ff449dac63816b5
e0687a1dcf735e661dfbce70da90a44fe9e82f6ecf2f192746186de286b3c6c9eb11f485bfc8eb2139e98d1796bf6119f4376eb95826043d4de08523fc3b709a
af5c79f5b640e94dfb74e72119e534f0ace076faad75b4ce13ee648281d35060a97461ae8a33b82608ac87500836a2a669a368959673128b24f72723725e53e9
5da40e77475b748f7c059777dc7dcc951818b661c9848e01d4a72bbde9af5bdc46df82ee656f000315c3f84a4c1e393088cad1d503e45a6977d7d7bcae315749
4c904d8ff20d0eb8e2c4dcbc8b294132a4e776195c460f1f4ea668bdc6922aeb93693fcf2c98088e469cc577306629aa59b084d3a7fce156bdd55001f50b55b3
04e9fc3c19aa1377a579208229b857cdce2901c1507d34d0e10ee86f02530a20ff51e6d70d2470edd34db9c8364d89ec590efa1787990391a454dbd7abe53817
fb69d607809a81fbc8bc051dd158b7634e00d56d50391acdd888bdde3bc0457d5e98bf49c375aef80a7bd37b54ae1078b3d2101a801581841f5fee0074a5db4f
574c507ac4a75ff152f776b294038869a50f8cd25c8d772a3a834a23ae0b658be368018d57a8c4e75120103b325d3aaed546742e6ab6be6707f68d29b40d5519
c5229522eb4485a419becd5fc41b8a53f5a4ace6387709e90c2a841a8f75a09f6cc471b67355bdceccae31a903d1a96ffcb9c8c9be89e5f8e852269554c2b563
96cd5e97c2f6222316daf97342c13a7d5a3b6c0474f11320c667976ee69d1068bc97de8a45291c63d78069e0ddea812e9d70f4f39ca16cc2132c6a71f6d9a3ce
63fda74cab409af0b8f0ca924c391e2c6ffda7219c5e75c3e481d7486fa23027915679316f1e2ed7e67d1ba03af41a3d23e2b3dc20c218c3e7aea0862cd3c1a0
f7b8064428e3fa4b5760d514def9304b1901828e7f26fe4dad1d1d67bf9d654928cd162ce508942bb5810a35595c9c90f380f0bacd4b7059c4ea608cc8f44f32
b4d3e7307c136eb7153d57a2a0147b09b5415e902a28fabbf10b422922680774500f7e0fbf9989425c514318f35f6747efd8b9dbe27927903371a24cd10ec81b
6cb5ca0743e30f46e8dc48eac367bc7929fcc6ef757a3c03ec14d3059808f3b8617babb1c7d391dafee049c51b92a2cacbabe38dd6fdefd826649686e06c4300
dcea9faf03535a75e79319cb3e401eaddb8b239bfb439057e6d1cfc343e92936a4ec8f85453db88ec47fe936c034a1245e4d6bbcaf6897b0a942888d1eaac86c
09a8dedcba27c4c5548f1ccbc6f77e67498c8ed964013156c0ceb57ed7d7579ee384f71632799241dd0b3677f93b27fea2a38265ec32020101b98c5fb3e67b5b
3bcc578c6c6e1a32e0308ae2229173a03313169464b6d1b389631f4f448658dc5a9d60a0c7c87a7459ad47efb634fd1104f89dd82504bc64a5bbaf83f591c3cf
0fb0a7d972afc0cdc4e6dccbd4abcdd7aaba458a1e9258f558a44b7757f7a94fac80e16c931a4f73f3ba1f22120eb9770a76b01e4b90e9b9d2099b92dc28fdd8
9be42a4bd46c25bdf926961eec52deb8098c4e01e397605ee64dd25437ee198fdfd95935762a5845c80529e81226be15af6ff8a2508d5969cdd6e867a61d0650
02b230b2d9df0a0b7aef08ac3a28be9a4c84af228adaaa27425b996bbfd8b18143352ce0ea2eff4c7a56031912526b41abffd390711a50a45a3e811dc0245862
317c906f07977e8aa5b09fdc045101cce53e4fb6db11b8bad021c90f8917064358b2a982c094c378bab6d4b78f1b3c4dec5822c523b91b9749efce2f15fbf895
8e61d1d22448f4f8cc15cf53fa45536ff556e25bd8a85323f40156934f8d3ebc65d7cc78a9372a5c4a63d57316c301c72443868e840c629487b6c2c554de4f24
5ca72122e102d35113c5ffe13853190ce18ce6282bc79d1f8e514a13b6f4b11e97e00ff8aeeb497b61d6365ca6d489584e96fc1069ef670f3b0fc1860b26e2a2
628b4a969ee000d8fd7027f7cd79b0140d7f430b1b79093cd1db79d9738882214b9843d32726124d6ac2f52585940d05738a237d0ce187317592b07880beffd7
dd787c6ebd3b4afb3badbefbb1f1c3c113afbe9b6a02ebe88f0105a3dd1a8fa7105f1450d047d2a1c6b4dbb738d80331755aacd0fb8b951c31e467540726ff1c
b3c3ac3f3800eee956c14d0413fc94d9ded99201e07106a7030b7f2828067b9b640b425e3a3f2edc09589b9d55f6e18a4f38cfed91049b87d19d3d1ca83c79c4
a8055db2ab63981172c75895135de4d709e4c0606826f657586e6d1e6ea2309751db10650f80370845a3f4b156ac0638760bdf5d94bc70015a3f6e4c1a1816a2
e72103d1b08736710390e8470d9f1a534410c6d612dd8c6e1aa2f77cef669b3d4ff37d63134d46445d323a3afe8e9c48cf8c9f00ce737c9ba737ca07146f020e7

2048 bit N, 2^{19} undiscoverability We now give parameters where exploitation is more practical but the backdoor is easier to discover. Compared to the previous parameters, the backdoor polynomial $f(x)$ has only 4 terms instead of 6, leading to a smaller lattice, but less entropy in the

choice of f. For these parameters we use modulus and output sizes corresponding to ISO's 112-bit security level.

$f(x) = x^{150} - x^{66} - x^{57} + x^{40}$

We decrease the degree of f to 150, so that the highest block of output needed is the 150th instead of the 2850th. However, we need $f(e) \geq \varphi(p)$ (since $f(e)$ must be divisible by $\varphi(p)$ and if it is zero there is no backdoor). So decreasing the degree of f requires us to use larger e. Here we use $e = 65537$, a commonly used exponent for RSA encryption.

Outputs are 1824 bits (so that the unknown part of each state is 224 bits, or twice the security level).

We use multiplicity $t = 3$ for Coppersmith, giving lattice dimension 70.

Exploitation took 3 core-minutes on an Intel Xeon E5-2699 v4 processor, using fplll.

q can be any 402-bit prime; p is the following 1646-bit prime such that $f(e) \equiv 0 \bmod \varphi(p)$:

```
2ba2ab9c8a742e6c463d8be9ebada976945325682bb788376bd30ce28c6df1913f3b0a9cfbb2c11f
8ec87e6e49a324ff9baa7756ad043085f01b343c676f97b8d684a4de8b5a4142ad22fb3acad3da5d
133ba2d002c32e2b8a5b3785b9176d4358dedf651d09aef345f3438bd91bd47971d7c1f0bf2326cc
3770b0c3e23678d5650619c1c03c1c0e59d8c5e4e5f3ba03d1a3e3cec0cd049fd9047e3b097f148f
3f5af11073a1fc6e0f3aa72072f061819aba80fba403adb6a09ffac1ead41bbc4d021cd98f17e508
34c10b7a8c81
```

1024-bit N, no claimed undiscoverability Finally, we give the small parameters we used as an example, with $e = 17$, and $e^{200} - e^{180} + e^{20} - 1 \equiv 0 \bmod \varphi(N)$. These parameters were generated for illustration purposes rather than with undiscoverability in mind.

```
N =
f17351a63a5b2042c08e423466bf654be5f59bcefd173ee4ba4d5c129439ed366da97ce83550e2a8
139bb93a909107060c755eedf3784c725e57f6207ae2a7a44dbdaa96e1db5c169a4a31a254329026
0cb9688e27f6f424d79f140e978f6117adad579a05b7ce76b932e5aff3c7461fdac9c08112e7f9e5
1a64e0ba949bcf3f
```

which factors as

```
p =
00027c0ef9a55007e7f9f77963a9fb5f6d87b4e39a83a4ac2ddccee72a70ddfeb8277ade35ca7ff4
4f9a96a191fdb7adda2508ab2a4b212a8e7e3cf43a4760df
```

```
q =
0000612dbd55fd70051f60effe91db4427fb1939872660e8d161f23c193be198a9dda8a6e6613634
f176a41cbd8f789ea23a2eb19d07bc0c2fe67d152e388fb862f57da1
```

Example verification code can be found in Appendix A.5.

A.4 $e = e_0 e_1^{-1}$ is insecure.

More generally, we consider exponents of the form $e \equiv e_0 e_1^{-1} \pmod{\varphi(N)}$ where e_0 and e_1 are small. Such an exponent remains efficient to compute for any e_0 and e_1 if the factorization of N is known.

Theorem 10. *Instantiating MS PRG with exponent $e \equiv e_0 e_1^{-1} \bmod \varphi(N)$ for e_0, e_1 small allows an efficient state recovery attack from one output for state size $r < n(1/(e_0 + \binom{e_1}{2}))$, in time polynomial in $\max(e_0, e_1)$ and $\lg N$.*

Security Parameter γ	n	n/r	Valid (e_0, e_1) pairs
80	1024	6.4	0
112	2048	9.14	2
128	3072	12	4
192	7680	20	13
256	15630	30	29

Table 1: **Weak exponent pairs permitted by ISO parameters.** We combine the bounds on efficient state recovery from Theorem 10 with the parameters dictated by the ISO standard to determine the number of vulnerable exponents for each security level.

Proof. This choice of e enables the construction of polynomial

$$f(s_0, s_1) = s_0^{e_0} - (2^k s_1 + b_1)^{e_1}.$$

This can be rewritten as $f(s_0, s_1) = s_0^{e_0} + \sum_{i=0}^{e_1} c_i s_1^i$, which leads to a Coppersmith lattice generated by the rows of basis matrix

$$B = \begin{bmatrix} R^{e_0} & R^{e_1} c_{e_1} & R^{e_1-1} c_{e_1-1} & \dots & c_0 \\ 0 & R^{e_1} N & 0 & \dots & 0 \\ 0 & 0 & R^{e_1-1} N & \dots & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & \dots & N \end{bmatrix}.$$

The dimension of this lattice is $e_1 + 2$ and the determinant is $N^{e_1+1} R^{e_0 + e_1(e_1+1)/2}$ where R is the bound on $|s_0|, |s_1|$. Heuristically, this allows recovery of s_0 and s_1 when $R < N^{1/(e_0 + e_1(e_1+1)/2)}$. This is equivalent to $e_0 + e_1(e_1+1)/2 < n/r$. □

We apply the bounds in the above theorem to the ISO parameters to see when a feasible attack exploiting this weak choice of exponent is permitted by the parameter choices.

Using this heuristic and requirements on the structure of e, we can determine the average number of valid (e_0, e_1) pairs leading to a distinct weak e for which Coppersmith's method succeeds to recover the RNG state. The results are given in Table 1.

It is possible to marginally improve the performance of Coppersmith's method in several ways, but this simple analysis suffices to demonstrate a few key points. First, the attack works in practice, and it even works when the largest modulus size is used. Second, there are relatively few exponents which may be weak with respect to this attack. Third, we will show that an attack of this form is *detectable*, meaning if an attacker persuades a victim to use such a weak exponent, the victim can efficiently determine if such an exponent is weak. Further, detection of a weak exponent yields the factorization of the modulus.

Theorem 11. *The special-form exponent backdoor is detectable. That is, given public modulus N and exponent e, it is efficient to determine if there exist small e_0, e_1 such that $e \equiv e_0 e_1^{-1} \pmod{\varphi(N)}$.*

Proof. To find e_0 and e_1 with high probability, we can simply do a brute force over pairs of small e_0, and e_1, and check whether $e \equiv e_0 e_1^{-1} \bmod \varphi(N)$. It is efficient because for the concrete parameters

proposed in the standard, the number of small (e_0, e_1) are limit enough to iterate through all of them. Additionally, once such an equivalence is known, we have $\varphi(N) \mid ee_1 - e_0$ and $ee_1 - e_0$ only slightly larger than $\varphi(N)$, so it is easy to determine $\varphi(N)$ and thus the factorization from e, e_0, e_1. □

Theorem 12. *The special-form exponent backdoor is not plausibly deniable. In other words, it is possible to distinguish between an exponent $e \in \mathbb{Z}_{\varphi(N)}$ generated randomly and $e = e_0 e_1^{-1}$ generated to conform to the backdoor.*

Proof. The proof of Theorem 11 gives a method to efficiently detect if e has the special form. To complete the argument, we show that such an exponent is overwhelmingly unlikely to have special form if generated uniformly at random. Since $e_0, e_1 \leq E$, there are at most $E^2 = O(\lg^2 N)$ special form exponents and approximately N valid exponents. For the parameter sizes used in Micali-Schnorr, this means we can detect whether the parameters were generated dishonestly with respect to this backdoor with a false negative rate of 0 and a false positive rate of less than 2^{-1000}. □

This investigation of Micali-Schnorr with weak exponents demonstrates that the ISO parameters are incomplete, and not all attacks have been eliminated from the standard. Although this attack is efficient, the disadvantage is that it is detectable, and would be difficult to perform in practice without the threat of accidentally exposing the factorization of N.

"Nothing up my sleeve" parameters. Although having a large exponent may be unusual, there is the possibility of it being disguised as a innocuous-seeming constant. The exponent for the following public key is based on the digits of $\pi = 3.14159\ldots$, but the inverse of e is small: $e \equiv 5^{-1} \pmod{\varphi(N)}$. Example verification code can be found in Appendix A.5. We leave it as an exercise to the reader to recover the factorization.

```
e =
31415926535897932384626433832795028841971693993751058209749445
92307816406286208998628034825342117067982148086513282306647093
84460955058223172535940812848111745028410270193852110555964462
29489549303819644288109756659334461284756482337867831652712019
09145648566923460348610454326648213393607260249141273724593646
90828205057904964694145265114203583480543017755295093967247853
14473853768601703608298900695818792485036015795124469751158648
34232981702282978024512185647195845547038058553635079877067449
32000141643103568465954665107904174028590478966056256766324482
95783971037101323127767011065093483627404478045559895421373

N =
39269908169872415480783042290993786052464617492188822762186807
40384770507857761248285043531677646334977685108141602883308867
30576193822778965669926016060139681285512837742315138194955577
86861936629774555360137195824168076605945602922334789565890023
86432060708654325435763067908310266742009075311426592155742102
19181870219019805590650376833502499649775088745000720744322712
39798910830524344272834113148801575679297398424503041238163201
69252588963664576543485275388222643231464070095285873656756143
03577976593282594799360871751228487780301961771829747487554943
87948667227434687425583498529809870418645678480508315970197
```

A.5 Example verification code

These scripts are also available at `https://github.com/ucsd-hacc/msdrbg_code`.

```
# Verify SUS Parameters
N = int("0x"
    "f17351a63a5b2042c08e423466bf654be5f59bcefd173ee4ba4d5c129439ed36"
    "6da97ce83550e2a8139bb93a909107060c755eedf3784c725e57f6207ae2a7a4"
    "4dbdaa96e1db5c169a4a31a2543290260cb9688e27f6f424d79f140e978f6117"
    "adad579a05b7ce76b932e5aff3c7461fdac9c08112e7f9e51a64e0ba949bcf3f",
    16)

p = int("0x"
    "00027c0ef9a55007e7f9f77963a9fb5f6d87b4e39a83a4ac2ddccee72a70ddfe"
    "b8277ade35ca7ff44f9a96a191fdb7adda2508ab2a4b212a8e7e3cf43a4760df",
    16)
q = int("0x"
    "0000612dbd55fd70051f60effe91db4427fb1939872660e8d161f23c193be198"
    "a9dda8a6e6613634f176a41cbd8f789ea23a2eb19d07bc0c2fe67d152e388fb8"
    "62f57da1",
    16)
e = 17

from math import gcd
phi_N = (p - 1) * (q - 1) // gcd(p-1, q-1)
val = pow(e, 200, phi_N) - pow(e, 180, phi_N) + pow(e, 20, phi_N) - 1
assert val % phi_N == 0

# Apply RSA PRG update function
import random
states = [random.randrange(N)]
for _ in range(200): states += [pow(states[-1], e, N)]

# Check relations
assert (states[200] * states[20] - states[180] * states[0]) % N == 0
print("The SUS parameters verify successfully")
```

Listing 1: Python script verifying our example parameters for the SUS attack.

```
# Verify eSP Parameters
N = int("0x"
    "fa4351292767d540d8dd2ef15b39b02e29b56ad2125add6964203eb0029ba7aa8d44c8e5"
    "65df7818f6eb959c37f83349ec4b1514ed42741b8e772028db779f5e362234b782b9064c"
    "ef07dead66bced100eb71cc8bf7f1325959eb304e90ecf6e53c4eff9708ebd3c7641ea2b"
    "aed50fbc679b4f0e06ad60b8e70c7836a5f99e1571be6194afe04c81a6eef8281bb43b99"
    "f17ee715e5f1a95e36aee8c4eb8757aa8ff108acdf29136cf543bb8831074b71fc63cc44"
    "4ca4d3d7811b6cdca33ec889dddc77828510ff0d10ac07b0f691ce2995a72137abf32816"
    "eed9cd322075e7450326ff2fba0b65c19c477d61407eccf48f5857b5ff95cf4c63702b9f"
    "09f9ca83", 16)
p = int("0x"
    "9da439c726a6c69087a81e49644d5e2d28937de5ba08c475a615813025318e2ae05f3174"
    "8f685c68f02b6883f10d407012970b50b28da780a6b157b7c80011a3857a877d56b5a8d4"
    "fc0a96bf62b7ec86d735de29628e230665356f535df7664723659f8f88847baaf9526393"
    "a521", 16)
q = int("0x"
    "1966956586a77b7e1fda36f722223a66b161f10027c53b2b39fde757e01fa6ef8b79e62e"
    "0b423546ebc2473011c3a79b4379788fff85eb743bfdbfe1d679511042295e5c6a5cf634"
    "b650bf83a997ecef1abef4467e02103de493f1f269bdb9088c6bc0f114971d05c983f499"
    "a48551bdeb901d3bc3290fbbf49368e5d9495b3922bdc03a9ec38e49aee6f0cedc78a1e2"
    "c5723", 16)
e = 5
c_q = 187
l = 504

assert pow(e, l, p - 1) == 1
assert pow(e, l, q - 1) == c_q

# Apply RSA PRG update function
import random
states = [random.randrange(N)]
for _ in range(l): states += [pow(states[-1], e, N)]

# Check relations
assert states[0] % p == states[l] % p
assert pow(states[0], c_q, q) == states[l] % q
print("The eSP parameters verify successfully")
```

Listing 2: Python script verifying our example parameters for the eSP attack.

```
# Verify nothing-up-my-sleeve Bad-e parameters
exponent_PI = int(
    "31415926535897932384626433832795028841971693993751058209749445"
    "92307816406286208998628034825342117067982148086513282306647093"
    "84460955058223172535940812848111745028410270193852110555964462"
    "29489549303819644288109756659334461284756482337867831652712019"
    "09145648566923460348610454326648213393607260249141273724593646"
    "90828205057904964694145265114203583480543017755295093967247853"
    "14473853768601703608298900695818792485036015795124469751158648"
    "34232981702282978024512185647195845547038058553635079877067449"
    "32000141643103568465954665107904174028590478966056256766324482"
    "95783971037101323127767011065093483627404478045559895421373")

# N is the product of two 1024-bit primes
N = int(
    "39269908169872415480783042290993786052464617492188822762186807"
    "40384770507857761248285043531677646334977685108141602883308867"
    "30576193822778965669926016060139681285512837742315138194955577"
    "86861936629774555360137195824168076605945602922334789565890023"
    "86432060708654325435763067908310266742009075311426592155742102"
    "19181870219019805590650376833502499649775088745000720744322712"
    "39798910830524344272834113148801575679297398424503041238163201"
    "69252588963664576543485275388222643231464070095285873656756143"
    "03577976593282594799360871751228487780301961771829747487554943"
    "87948667227434687425583498529809870418645678480508315970197")

# The decryption exponent corresponding to the public exponent is 5.
import random
p = random.randrange(N)
c = pow(p, exponent_PI, N)
p2 = pow(c, 5, N)

assert p == p2
print("The nothing-up-my-sleeve parameters verify successfully.")
```

Listing 3: Python script verifying our "Nothing up my sleeve" parameters for the Bad-e attack.