

Cryptographic Engineering

working draft

Matthew D. Green
Johns Hopkins University

Excerpt: Chapter Chapter 14, Random number generation

§14.3.2 ANSI X9.31 PRG; Debian's OpenSSL PRG; factorable RSA keys (accidental failures; Dual EC is the Nov 30 reading) (book pages 31-32)

Posted for 601.445/601.645 Practical Cryptographic Systems, Fall 2026
Lecture 17, Wed Oct 28: implementation failures

<https://practicalcrypto.org>

This is an excerpt from an unfinished book draft: expect gaps, rough edges, and errors, and note that section numbers and page numbers may change in later drafts. Cross-references to other chapters point to material that is not included here. Copyright © 2026 Matthew D. Green. All rights reserved. Posted for the students of the course above; **please do not redistribute** — link to the course site instead. Built 2026-09-06.

14.3.2 ANSI X9.31 PRG

Juniper Netscreen hardware is not the only mass-market VPN device that has been known to contain a passively-exploitable RNG flaw. In 2017 a related flaw was discovered in Fortinet devices.

Prior to the release of NIST SP 800-90A, many FIPS-certified devices made use of a now-deprecated pseudorandom generator, which was standardized by the ANSI X9.31 committee in 1998. The ANSI X9.31 generator pre-dates the development of NIST SP 800-90A [33], and contains a much more vulnerable design. The ANSI design makes use of a block cipher, typically either 3DES (in earlier implementations) or AES. Figure 14.15 shows one round of the generation process. There are two “seeds” in this design: a fixed key K that does not change between rounds, and a “state” (seed) V that is initially set to a random value, but is updated within each round of generation. The generator also takes as input a unique “timestamp” that can be drawn from a system clock, or can simply be an ascending counter that does not repeat.

For nearly as long as ANSI X9.31 has existed, researchers have known that it was vulnerable to a specific attack: if an attacker can learn the value K and guess the timestamp T_i , then the generator’s internal state can be recovered from seeing a single block of output R_i [186]. With further ability to guess timestamp values, the attacker can then wind the generator both forwards *and* backwards to recover future and past generator outputs!

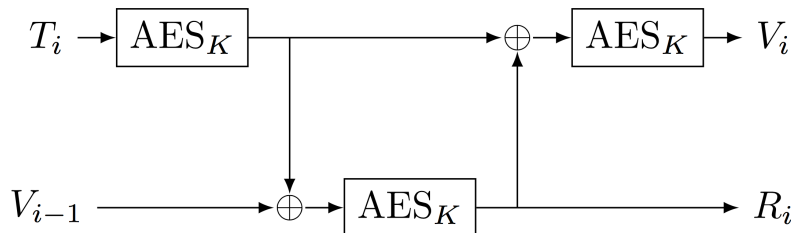


Figure 14.15: PRG generation loop for the ANSI X9.31 generator, here implemented using AES. T_i is a “timestamp” value (in practice, this can simply be an incrementing counter). V_{i-1} and V_i represent the seed/state, which must be initialized to a random value at V_0 and will update at each iteration. K is a fixed AES key that does not change or update. R_i represents a block of random output. All values are strings of the form $\{0, 1\}^b$ where b is the cipher’s block size (128 bits in this example.) Source: [88].

Of course, this requires an apparently strong assumption: that the key K is known to the attacker. Unfortunately, during the time ANSI X9.31 was allowed in devices, the U.S. FIPS standards allowed many vendors to use a *fixed and hard-coded* value of K in devices, which meant that K could potentially be recovered through simple reverse-engineering. Even

worse, a large class of Fortinet firewall/VPN devices using software released from 2009–2014 included a hard-coded key drawn from public FIPS test vectors, which made tens of thousands of devices vulnerable to state-recovery attacks and TLS/IPsec decryption [88].

It is quite remarkable to note that two major U.S. VPN device vendors, Juniper/Netscreen and Fortinet, both contained critical RNG vulnerabilities during the same time period, both of which enabled efficient decryption of VPN connections. One might even call it bad luck.

Debian’s OpenSSL PRG

A challenge with RNG implementations is that it is relatively easy to break them in ways that are hard to detect. Back in 2008, a version of OpenSSL that shipped with the Debian distribution of Linux was found to contain a subtle flaw that resulted in the generation of thousands of low-entropy (but apparently valid) SSH and TLS/SSL keys [284]. The accidental flaw (CVE-2008-0166) was ironically the result of an attempt to improve the security of OpenSSL’s code.

The issue in this version of OpenSSL was triggered by the fact that the library’s internal PRG used a large quantity of seed material to initialize the PRG, some of which included sections of *uninitialized memory*. This was not by itself a security vulnerability, since the remainder of the seed material was high-entropy and sufficient to properly seed the generator. (Indeed, the use of uninitialized memory was believed by the OpenSSL developers to be a harmless, even possibly helpful element, since this memory might contain additional entropy beyond the normal seed material.) Unfortunately, during a security analysis, a developer using the Valgrind memory-safety analysis tool detected the use of uninitialized memory, and flagged it as a potential security vulnerability [235]. The “fix” was to disable specific lines of code that were, unfortunately, essential to ensuring that the remainder of the seed material was properly hashed into the OpenSSL PRG seed.

The consequence was that the only effective source of unpredictable material in this generator was the *process ID* (`pid`). With the usual process-ID limit, this meant that there were only 32,768 possible values for this variable, the result was that the OpenSSL seed could be easily brute-forced on some architectures. Unfortunately, since the PRG itself is cryptographic, the resulting output bits appeared indistinguishable from properly-working PRG output to simple manual inspection. Even worse: command-line tools for generating keys were short-lived programs and hence seeded the PRG immediately before generating TLS/SSL keypairs, which meant that there could truly be at most tens of thousands of possible keypairs that could be produced in the worst case. Luciano Bello discovered the flaw, which Debian disclosed in May 2008. Fixing the software did not repair keys already generated by affected versions: those keys needed to be replaced, and certificates containing compromised public keys needed to be revoked.