Technical Perspective
Backdoor Engineering

Communications of the ACM, Vol. 61, No. 11, November 2018, p. 147, DOI:10.1145/3266289

By Markus G. Kuhn

IMAGINE YOU ARE a cyber spy. Your day job is to tap cryptographically protected communications systems. But how? Straightforward cryptanalysis has long become impractical: the task of breaking modern algorithms far exceeds all computational power available to humanity. That leaves *sabotage*.

You can target many Achilles heels of a crypto system: random-bit generators, side channels, binary builds, certification authorities, weak default configurations. You infiltrate the teams that design, implement and standardize commercial security systems and plant there hidden weaknesses, known as *backdoors*, that later allow you to bypass the cryptography.

Take random-bit generation. Security protocols distinguish intended peers from intruders only through their knowledge of secret bit sequences. Servers have to chose many key values at random to protect each communication session, and an adversary who can successfully guess these can impersonate legitimate users.

One trick to backdoor a random generator can be understood with basic high-school algebra. A deterministic random-bit generator (DRBG) is initialized (seeded) with a start state s_0, and then iterated with some generator function: $s_{i+1} := G(s_i)$

$$s_0 \xrightarrow{G(s_0)} s_1 \xrightarrow{G(s_1)} s_2 \quad \cdots$$

In simple DRBGs (say for simulations), the s_i may serve as both the state of the generator as well as its output. So anyone who saw an output s_i and knows G can easily predict all future outputs. Crypto-grade DRBGs make four improvements: (a) hardware noise sources (slow) seed s_0, (b) the state s_i has hundreds or thousands of bits, (c) a second function H derives output values $r_i := H(s_i)$

$$\begin{array}{ccccccc} s_0 & \xrightarrow{G(s_0)} & s_1 & \xrightarrow{G(s_1)} & s_2 & \cdots \\ \downarrow{\scriptstyle H(s_0)} & & \downarrow{\scriptstyle H(s_1)} & & \downarrow{\scriptstyle H(s_2)} & \\ r_0 & & r_1 & & r_2 & \end{array}$$

and (d) both G and H are *one-way functions*. These can be computed efficiently, but their inverses not. After H, an adversary who can see some of the outputs r_i cannot infer anything about the internal states s_i or other outputs r_j. We know many excellent choices for G and H: one-way functions carefully engineered to be fast and to have no other known exploitable properties. Most are constructed from secure hash functions or block ciphers.

As a saboteur, you do not want these used. Instead, you lure your victims towards a far more dangerous option: the class of algebraic one-way functions that enabled public-key crypto. These are orders of magnitude slower and require much bigger values for equal security. Modular exponentiation is a simple example. If you follow a few rules for choosing a big integer g and a big prime number p, then $G(x) := g^x \bmod p$ is such a one-way function. While g^x alone is monotonic, and thus easy to invert, the mod p operation (take the remainder after division by p) ensures that the result remains uniformly spread over a fixed interval and appears to behave highly randomly. The inverse *discrete logarithm* problem, of calculating x when given $(g^x \bmod p, p, g)$, becomes computationally infeasible. (In the following, we drop mention of the mod p operation, and just apply it automatically after each arithmetic operation.) The exponentiation operator g^x has an important additional property, not affected by the mod operation: $(g^x)^y = (g^y)^x$. While this commutativity is completely useless to honest designers of DRBGs, it can be invaluable to saboteurs.

Convince your victims that $G(s_i) := g^{s_i}$ and $H(s_i) := h^{s_i}$ are excellent choices for generating random numbers of the highest security:

$$\begin{array}{ccccccc} s_0 & \xrightarrow{g^{s_0}} & s_1 & \xrightarrow{g^{s_1}} & s_2 & \cdots \\ \downarrow{\scriptstyle h^{s_0}} & & \downarrow{\scriptstyle h^{s_1}} & & \downarrow{\scriptstyle h^{s_2}} & \\ r_0 & & r_1 & & r_2 & \end{array}$$

You can claim "provable security based on number-theoretical assumptions", but this is, of course, just a smoke screen. The sole advantage of this construction is that it allows a backdoor. If you can choose g as $g := h^e$, then knowing your secret integer e immediately allows you to convert any output value r_i into the next internal state of the DRBG as $(r_i)^e = (h^{s_i})^e = (h^e)^{s_i} = g^{s_i} = s_{i+1}$:

$$\begin{array}{ccccccc} s_0 & \xrightarrow{g^{s_0}} & s_1 & \xrightarrow{g^{s_1}} & s_2 & \cdots \\ \downarrow{\scriptstyle h^{s_0}} \nearrow{\scriptstyle r_0^e} & & \downarrow{\scriptstyle h^{s_1}} \nearrow{\scriptstyle r_1^e} & & \downarrow{\scriptstyle h^{s_2}} & \\ r_0 & & r_1 & & r_2 & \end{array}$$

So if you contact a server and receive one r_i, you can now immediately predict all future r_j used to protect the communication with others, and decrypt or impersonate their messages. Job done. And nobody else can do this, because finding e from h and g is computationally infeasible (the aforementioned *discrete logarithm* problem). Unless, of course, they steal your backdoor by generating their own e' and replacing your g with their $g' := h^{e'}$.

The following article by Checkoway et al. reports on the amazing independent reconstruction of exactly such a backdoor, discovered in the firmware of a VPN router commonly used to secure access to corporate intranets. In 2004, the NSA planted the above DRBG in NIST standard SP 800-90, including a g and h of their choice. The details differ only slightly (elliptic curve operations rather than modular exponentiation, which uses slightly different notation; the top 16 bits of r_i discarded, can be guessed via trial and error). The basic idea is identical.

But planting a backdoor in a standard is not enough. You now also have to ensure industry implements it correctly, such that an r_i reaches you intact. And that nobody else replaces your g. And that is where this story begins.

Markus G. Kuhn (mgk25@cam.ac.uk) is a Senior Lecturer teaching computer security and cryptography at the University of Cambridge, England.