from *SIAM News*, Volume 34, Number 2

# Safe Against Cycling: Researchers Confirm Invulnerability Of RSA Cryptosystem

*By Barry A. Cipra*

The Achilles heel of public-key cryptography is the unproved assumption that problems like factoring large numbers are really as hard as they appear. But unlike the Greek warrior, cryptographers have to worry about other potentially vulnerable spots as well. For the number-theoretic RSA cryptosystem, a long-standing concern was a procedure known as the cycling attack. But that's no longer the case. John Friedlander of the University of Toronto, Carl Pomerance of Bell Labs, and Igor Shparlinski of Macquarie University, in Sydney, Australia, have shown that RSA is impervious to the cycling attack procedure.

That doesn't mean the attack *never* works—in codebreaking, you can always get lucky. But the likelihood of success, the researchers have shown, is asymptotically nil, which means that almost all randomly selected RSA systems are safe against cycling. Their results also indicate the workability of a new technique called timed-release cryptography, in which the key to a secret message is revealed only at the end of a time-consuming computational odyssey.

The number-theoretic essence of RSA resides in the formula

$$C = M^e \mod N,$$

where $C$ is the coded version of the "plaintext" message $M$, $e$ is a more or less arbitrary exponent, and $N = pq$ is the product of two (secret) primes $p$ and $q$. The computation of $C$ can be done in $O(\log e)$ steps, by repeated squaring—e.g., $M^8 = ((M^2)^2)^2$. The decoding computation is similar: $M = C^d \bmod N$, where $d$ is a secret number satisfying the condition $ed = 1 \bmod \phi(N)$, where $\phi(N) = (p - 1)(q - 1)$. The only mathematical restriction on $e$ (imposed to ensure the existence of a corresponding $d$) is that $e$ be relatively prime to $\phi(N)$. But even that can be lifted if all you want to do is use the formula $M^e \bmod N$ to produce pseudorandom numbers.

If you (foolishly) use $e = 1$ for your coding or randomization exponent, the math works fine, but there's clearly no secret coding or randomization going on. Any larger exponent, even $e = 2$, will, in general, produce a $C$ bearing no resemblance to $M$. But are larger exponents necessarily any safer?

The cycling attack bets that they're not. This approach computes a sequence of numbers by the iterative procedure $C_{n+1} = C_n^e \bmod N$, with $C_1 = C$, declaring victory when $C_{n+1} = C_1$—because that means $C_n = M$. Elementary number theory guarantees the existence of a "cycling value" $n$. (For exponents like $e = 2$, which aren't relatively prime to $\phi(N)$, the procedure doesn't necessarily return to $C_1$, but it does return to *some* earlier value. In this case, the cycling attack tries to expose the foolhardiness of using repeated exponentiation as a source of pseudorandom numbers.) But the attack will be successful only if $n$ is fairly small. Otherwise, you'd run out of time—the universe could die while you were still running cycles.

*The cycling attack does no better (it does, in fact, worse) than a straightforward effort to factor the number* N.

In showing that the cycling attack rarely works, Friedlander and colleagues confirmed a long held assumption. Their main result is that for almost all choices of $p$, $q$, $e$, and $M$, the smallest cycling value is nearly as large as $N$ itself. If $e$ is set equal to 2—a special case first investigated by Lenore and Manuel Blum and Michael Shub in a 1986 paper in *SIAM Journal on Computing*—the guarantee is weaker but still adequate: With relatively few exceptions, the smallest $n$ is nearly $N^{1/2}$. (A paper by Friedlander et al., slated for publication in *Mathematics of Computation*, contains the exact statements and complete proofs.) In short, the cycling attack does no better (it does, in fact, worse) than a straightforward effort to factor the number $N$.

The theorists' RSA cycling-proof proof also puts some rigor behind the new wrinkle in secure communication called timed-release cryptography. Proposed in 1996 by cryptographers Ronald Rivest of MIT and Adi Shamir of the Weizmann Institute, in Rehovot, Israel (the "R" and "S" of RSA), and David Wagner of the University of California, Berkeley, the new idea is to send messages that can be decoded only after the receiver has done a lengthy, time-consuming computation. The cryptographers have posted a challenge example at http://www.lcs.mit.edu/research/demos/cryptopuzzle0499. They estimate that the computation will take 35 years to complete, even allowing for future Moore's law doublings of computer power. (Most of the computation, they figure, will be done in the last few years.)

The time-consuming computation is to raise a given number, say $a$, to a given power, say $e$, a large number of times, say $k$, reducing mod $N$ along the way —i.e., compute $A = a^{e^k} \bmod N$. The sender can do this in $O(\log k)$ steps by computing $f = e^k \bmod \phi(N)$ (using repeated squaring) and then computing $a^f \bmod N$. But the receiver seems to have no recourse but to do all $k$ exponentiations. Even parallel processing seems to offer no help (or if it could, no one's figured out how).

There are various ways to use the resulting number $A$ to disguise a message $M$. The simplest is to let $C = M + A \bmod N$ and then send the 5-tuple $(C, N, a, e, k)$. Rivest et al. prefer coding $M$ with a key $K$ and then sending the 6-tuple $(C, L, N, a, e, k)$, where $L = K + A \bmod N$. In either case, the duration of the secret is $O(k)$ (but, of course, the constant depends on the receiver's computational power).

The theoretical sticking point for timed-release cryptography was the prospect that the computation $a^{e^n} \mod N$ for $n = 1, 2, \ldots$, might become periodic early on. If that were to happen, the time-consuming computation would suddenly be short-circuited. But that, the new results guarantee, is extremely unlikely. Even if $e$ is fixed at 2, the sequence $a^e, a^{e^2}, a^{e^3}, \ldots \pmod N$, will, with rare exceptions, have an extremely long period.

The cycling attack will continue to pester cryptographers—it's a potential threat for any new technique. But when it comes to codes based on number theory, cycling, it seems, is about as effective as pedaling a Schwinn to chase a Porsche.

*Barry A. Cipra is a mathematician and writer based in Northfield, Minnesota.*

## Win Some, Lose Some. . .

RSA may be safe, at least against the cycling attack (see accompanying article). But an algorithm with a similar-sounding acronym has recently found itself on potentially shaky ground: Daniel Bleichenbacher, a computer scientist at Bell Labs, has spotted a weakness in DSA. The problem is easy to fix, but it's a cautionary tale nonetheless.

Designed by the National Security Agency, DSA (Digital Signature Algorithm) is one of three algorithms approved for generating and verifying digital signatures under the Digital Signature Standard. This standard was developed by the National Institute of Standards and Technology and has been adopted by both the American National Standards Institute (ANSI) and the Institute of Electrical and Electronics Engineers (IEEE), among others.

DSA doesn't encrypt messages. All it does is append a string of bits computed from the message being sent, along with a private key belonging to the sender; using the sender's public key, the receiver can verify that the message was indeed sent by the owner of the private key.

Mathematically, DSA uses a large prime $p$ of between 512 and 1024 bits, another, 160-bit prime $q$ that divides $p - 1$, a number $g$ of order $q$ in the multiplicative group mod $p$ (easily obtained by letting $g = h^{(p-1)/q} \mod p$ with $1 < h < p - 1$ and testing that $g \neq 1$), and two private numbers $x$ and $k$ in the range from 1 to $q$. The $x$ is the sender's private key; the public key is $y = g^x \mod p$. (The security of $x$ is based on the presumed difficulty of computing discrete logarithms.) The $k$ is a secondary number that the sender must change with each message.

The signature to a message $M$ consists of two 160-bit numbers: $r = (g^k \mod p) \mod q$ and $s = k^{-1}(\text{SHA-1}(M) + rx) \mod q$, where SHA-1 is a specific function known as the Secure Hash Algorithm, which turns arbitrary-length bit strings into more or less random-looking 160-bit numbers. (The secure hash function is called "SHA-1" because NIST made a slight change in the original SHA after NSA had found a small flaw in it.) To verify a signature, the receiver computes $u_1 = \text{SHA-1}(M)s^{-1} \mod q$ and $u_2 = rs^{-1} \mod q$ and then checks whether $(g^{u_1} y^{u_2} \mod p) \mod q$ reproduces $r$; if it does, the receiver can be confident as to the source of the digitally signed message.

It's important that the sender never reuse a value of $k$ (which the receiver would spot as a reused $r$), because the receiver (or anyone else) could immediately compute the twice-used value $k$ from the difference $s - s' = k^{-1}(\text{SHA-1}(M) - \text{SHA-1}(M'))) \mod q$ and then obtain the private key $x$. But it's necessary to be even more careful than that. Cryptographers have shown that knowing just a couple of bits of each $k$ that a sender uses makes it possible to unravel $x$. So it's important for the bits of $k$ to be unpredictable.

DSA specifies a handful of pseudorandom number generators for the production of $k$s. One of them comes from an ANSI standard: Use the Secure Hash Algorithm to produce an initial, 160-bit number $k_0$, and then reduce it mod $q$.

That's where the trouble lies.

The problem, Bleichenbacher explains, is that the resulting $k$s are not uniformly distributed between 0 and $q - 1$—smaller numbers are twice as likely to occur as larger ones. This is easy to see from a smaller example: When reducing 5-bit numbers (from 0 to 31) mod $q = 23$, the numbers 0 through 8 each occur twice while the numbers 9 through 22 each occur just once. (The probabilistic conclusion assumes that the 160-bit numbers are generated with a uniform distribution. This seems to be the case for the Secure Hash Algorithm.)

In information-theoretic terms, the degree of nonuniformity in the ANSI random number generator is about the equivalent of knowing a little under a tenth of a digit of $k$. (To be exact, that much information is available only in the best case—"best," that is, for the cryptanalyst. An implementor may be lucky and choose the DSA parameters such that less information about the secret keys leaks.) What Bleichenbacher has done is to parlay this into a statistically based algorithm that takes roughly 4 million ($2^{22}$) signed messages, does a few million trillion ($2^{64}$) operations using several terabytes of memory, and spits out the no-longer-private key $x$.

Those numbers, especially the million trillion and the terabytes, mean that DSA is perfectly safe—for now. "While e-commerce is not currently threatened," Bleichenbacher says, "a good crypto-system should always have a comfortable security margin. That is, it should be secure even in 10 or 20 years from the day it is used, assuming the usual progress in hardware development. Without a fix, DSA would not have that security margin."

Fortunately, it looks as if the weakness exploited by Bleichenbacher's algorithm will be easy to fix. NIST is now preparing a revision of the specification. "In the meantime," says Edward Roback, chief of the Computer Security Division in NIST's Information Technology Laboratory, "those who are using DSA can continue to use it with confidence that DSA signatures done under the present standard will remain secure for many more years."

For Bleichenbacher, his algorithm "shows once again that, in order to get good standards, you have to publish them and make the standards openly available." He applauds NIST for doing exactly that. When it comes to secure computing, he says, "it would be bad for any organization to assume that it can generate good protocols without the help of the community." Even when that "help" comes in the form of an attack.—*Barry Cipra*