Design and implementation of a public-key signature system

D. J. Bernstein
University of Illinois at Chicago

$$\text{Fix } \alpha = \begin{array}{l}216443395665790188244691830394598492\,2\\ 644695550112583910720148205471118482\,1\\ 804953527132285055906448390746603282\,3\\ 156808412897607112435284308268998026\,8\\ 260461870329544947902850157399396179\,1\\ 488205972383346264832397985356295141\,3\end{array}.$$

A **secret key** is a pair of prime numbers (p, q) with $p \bmod 40 = 3$, $q \bmod 40 = 7$, $4 \cdot 2^{765} < p < 5 \cdot 2^{765}$, and $2^{800}\alpha < pq < 2^{800}(\alpha + 1)$.

Many people have realized that one can save space for public keys in an RSA-type system by prespecifying some leading or trailing bits. The idea was published in 1991 (EUROCRYPT '90, page 467). More historical details have been collected by Arjen Lenstra (ASIACRYPT '98, page 1). In 2000, three days before my talk, Vanstone and Zuccherato received US patent 6134325 on the same idea. Their patent application was in 1995, more than a year after the idea appeared in a printed publication, so the patent is invalid under 35 USC 102(b). I was planning to discuss this patent in my talk, but after a series of questions I decided that I didn't have the time.

Public key pq.

$2^{1535} < pq < 2^{1536}$.

Compress to pq mod 2^{800}.

Conjecturally: NFS computes $pq \mapsto p, q$ in $\approx 2^{100}$ steps.

ECM computes $pq \mapsto p, q$ with chance 2^{-50} in $\approx 2^{70}$ steps.

For $p > 0$ with $p \bmod 10 \in \{3, 7\}$:

Images of $2^{(p-1)/2}$ and $x^{(p+1)/2}$
in $(\mathbf{Z}/p)[x]/(x^2 - 3x + 1)$
are in $\{-1, 1\}$ if p is prime.

In practice, only if p is prime.

A **message** is a byte string.

A **hash value** is an integer h with $0 < h < 2^{1531}$, $h \bmod 8 = 1$.

A **hash function** is a function from $\{0,1\}^{256} \times \{\text{messages}\}$ to $\{\text{hash values}\}$.

For public key n,
hash function A, message m:

(r, h, f, s, t) is an
A-**signature of m under n**
if $r \in \{0,1\}^{256}$, $h = A(r,m)$,
$0 \le s < 2^{1536}$, $0 \le t < 2^{1536}$,
$f \in \{-2,-1,1,2\}$, $s^2 = tn + fh$.

Recap of the signature systems I mentioned in my talk:
$s^e \bmod n = m$: The RSA system. Trivially breakable. Slow verification.
$s^3 \bmod n = m$: Often incorrectly called the RSA system. Trivially breakable.
$s^2 \bmod n = m$: Often incorrectly called the Rabin system. Trivially breakable.
$s^2 \bmod n = A(r,m)$: The Rabin system. Unbroken.
$s^2 \bmod n = fA(r,m)$: The Rabin-Williams system. Unbroken.
$s^2 - tn = fA(r,m)$: The RWB system. Unbroken.

Can compute $s^2 - tn - fh$,
check if result is 0.

Faster: Reduce $s^2 - tn - fh$
modulo a secret prime ℓ with
$2^{114} < \ell < 2^{115}$, $\ell \bmod 5 \in \{2, 3\}$;
check if result is 0.

Chance $< 2^{-100}$ of error
for uniform random ℓ.

Given small $n_0, n_1, \ldots, n_{24}$ with $n = 2^{800}\alpha + \sum_j 2^{32j} n_j$:

$$n \equiv (2^{800}\alpha \bmod \ell) + \sum_j (2^{32j} \bmod \ell) n_j.$$

Precompute $2^{800}\alpha \bmod \ell$, etc.

Similarly s, t, h.

449 bytes for (r, h, f, s, t)
if h can be stored in 32 bytes.

257 bytes for (r, h, f, s).
Recover t as $(s^2 - fh)/n$.
Combine $\mathbf{Q}_2$ division, $\mathbf{R}$ division.

Could compute $s^2 \bmod n$,
compare to fh;
but faster to recover t and use ℓ.

The idea of combining 2-adic division with 0-adic division, to halve the time for exact division of small numbers, was published by Jebelean in 1993.

Signer must always generate **standard signatures**: s is a square modulo n; $s < n$; f is earliest in $1, 2, -1, -2$ with fh a square modulo n.

Signer should choose uniform random r for each signature.

To compute (f, s) given h, p, q:

$x = h^{(q+1)/4} \bmod q$.

$e = 1$ if $x^2 \equiv h \pmod q$, else -1.

$y = (eh)^{(p+1)/4} \bmod p$.

$f = e$ if $y^2 \equiv eh \pmod p$, else $2e$.

$x' = (f/e)^{(q+1)/4} x \bmod q$.

$y' = (f/e)^{(p+1)/4} y \bmod p$.

$s = x' + q(q^{p-2}(y' - x') \bmod p)$.

I neglected to mention in my talk that this procedure, without much programming effort, takes constant time on typical computers, so there is no risk from timing attacks.

Attacker has random public key n and oracles for $r, m \mapsto A(r, m)$, $m \mapsto$ uniform random standard A-signature of m under n.

Attacker's **success chance for** A is chance that attacker's output is (m, an A-signature for m) for some message m never given to the second oracle.

If attacker does $< 2^{80}$ queries
and has success chance ϵ
on average over all A
then in about the same time
can find square roots modulo n
with probability $\geq \epsilon - 2^{-98}$.

Hence can factor n
with probability $\geq (\epsilon - 2^{-98})/2$.

This type of theorem prevents stupid mistakes, but it doesn't prevent malice. I mentioned in my talk that one can easily construct a system where every computable hash function is insecure even though the same type of theorem holds; one can, for example, accept a program as a signature if the program produces the same result as the hash function on several inputs selected randomly by the verifier.

Given n, y square unit modulo n,
to find a square root of y:
Use attacker with fake oracles.

Hashing oracle generates
uniform random hash values h
and auxiliary (w, f) with
$fh \equiv w^2 y \pmod{n}$, proper f.

Signing oracle, given m,
generates uniform random r
and a standard signature with
a uniform random hash value.

Fails if (r, m) was already
assigned a hash value
by the hashing oracle.

Reasonable choice of A:

Fix $\beta =$
$$\begin{array}{l}
9070023117376067920656283394\\
4025700038548313316260683816\\
7207155960299447770935701734\\
2445432473581147748610084622\\
8086606416744290576148843994\\
1980451207039781438375110910\\
1525913570889283323670943497\\
2682323183703659506259243300\\
9275340926695314718129950300\\
2391936647247246615258712831\\
7549574535303667042772676966\\
9475959996390742757794266253\\
1747820635325409548281828172.
\end{array}$$

$$A(r, m) = 2^{320}\beta + 2^{64}z + 1$$

where $z = \text{SHA-256L}(r, m)$.

In my talk I alluded to a selective signature forgery method whose complexity is roughly the square root of the number of hash outputs. The attacker first obtains legitimate signatures on many messages, and sorts those signatures by h; then the attacker hashes many potential forgeries, looking for a match in h.

http://cr.yp.to
/papers/sigs.dvi

sigs-subscribe
@list.cr.yp.to