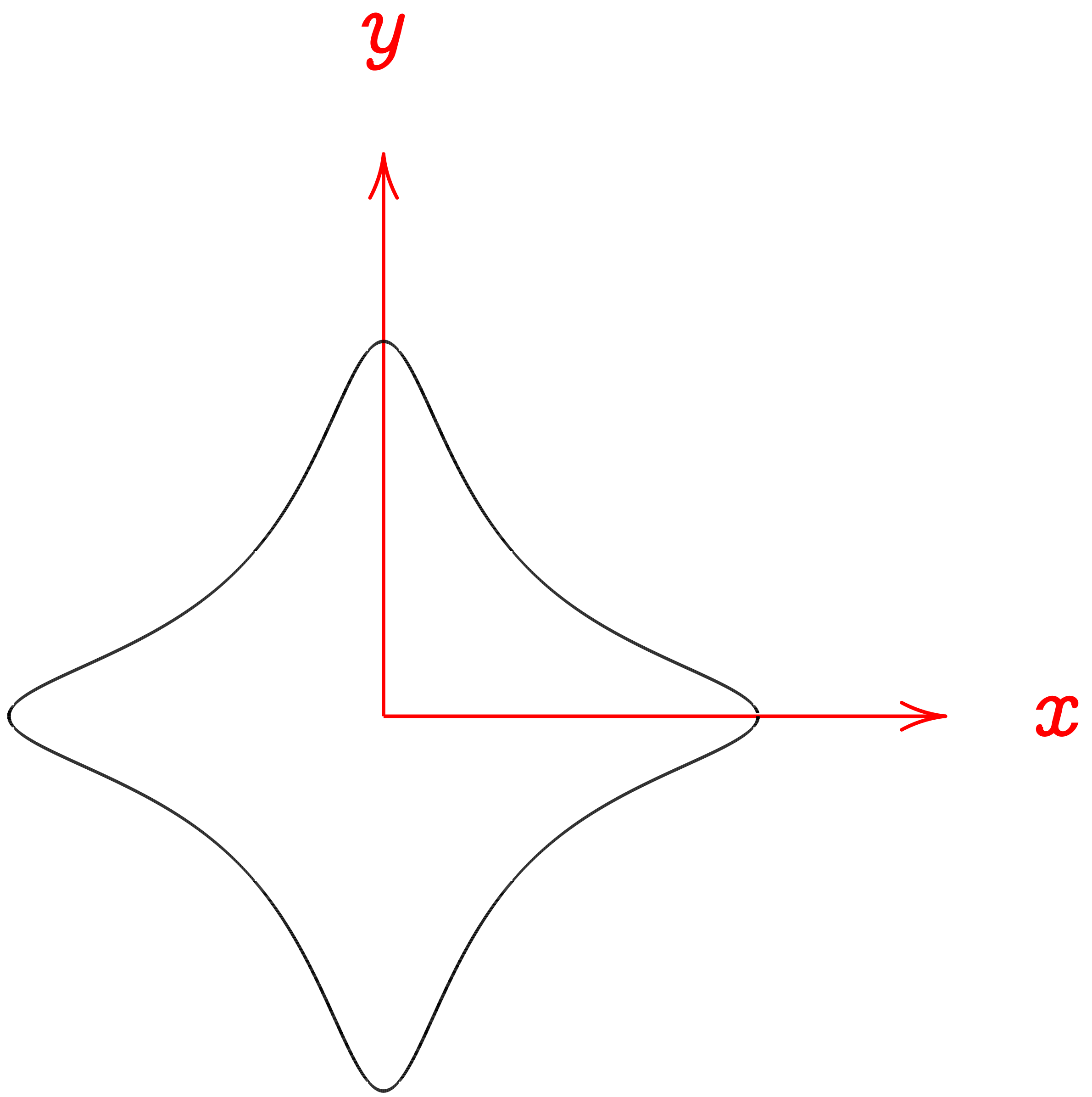
High-speed cryptography

D. J. Bernstein
University of Illinois at Chicago

NSF ITR–0716498



1. The Domain Name System

Mail sender at `snu.ac.kr`

DNS packet:
"The mail server for
`icisc.org`
has IP address
211.202.2.8."

Administrator at `icisc.org`

Now `snu.ac.kr` sends mail
to 211.202.2.8.

Is this system secure?

Many security holes
in DNS software:
BIND libresolv buffer overflow,
Microsoft cache promiscuity,
BIND 8 TSIG buffer overflow,
BIND 9 dig promiscuity, etc.

Fix: Use better DNS software.
`cr.yp.to/djbdns.html`

But what about protocol holes?

Stealing mail by attacking DNS

Mail sender at `snu.ac.kr`

DNS packet:
"The mail server for
`icisc.org`
has IP address
131.193.36.27."

Attacker anywhere on network

Now `snu.ac.kr` sends mail
to 131.193.36.27.

Real `icisc.org` never sees it.

No warning to `snu.ac.kr`.

Are attacks really so easy?

Can attacker guess
where mail is being sent?

Can attacker guess
time when mail is being sent?

Can attacker guess
UDP port for DNS packet?

Can attacker guess
the random 16-bit ID
that the mail sender
puts into its DNS request?

For sniffing attackers, yes; but
attackers anywhere on network?

July 2007: Emergency security update for BIND to change ID generation.

Previous ID generator was cryptanalyzed by Amit Klein: "This is a weak version (since the output is 16 bits, as opposed to the traditional 1 bit) of the ... mutually clock controlled (LFSR) generator ..."

Attacker legitimately receives 13 successive IDs from sender, reconstructs stream-cipher state, predicts sender's subsequent IDs.

Add signatures to DNS?

Long IDs and strong generators
don't stop sniffing attackers.

Obvious solution:
Public-key signatures in packets.

But many deployment obstacles:
many DNS implementations;
many different databases;
tiny packets, 512 bytes;
heavily loaded senders;
heavily loaded receivers.

Current Internet situation:
0% of DNS packets are signed.

Can change DNS-security protocol
to minimize effects on
implementations, databases.

But still need extremely small,
extremely fast signatures with
extremely fast verification.

For fastest verification:
state-of-the-art Rabin-Williams.
But that could be trouble
for signature time, space.

Let's look at some alternatives.

2. Secure authenticators

Standardize a prime $p = 1000003$.

Sender rolls 10-sided die
to generate independent
uniform random secrets
$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\ldots$,
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\ldots$,
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Sender meets receiver in private and tells receiver the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an **authenticator**

$$(m_{n,1}r_1 + \cdots + m_{n,5}r_5 \bmod p)$$
$$+ s_n \bmod 1000000$$

and the message number n.

e.g. $r_1 = 314159$, $r_2 = 265358$,
$r_3 = 979323$, $r_4 = 846264$,
$r_5 = 338327$, $s_{10} = 950288$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6r_1 + 7r_2 \bmod p)$
$\quad + s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 265358$
$\quad \bmod 1000003)$
$\quad + 950288 \bmod 1000000 =$
$742451 + 950288 \bmod 1000000 =$
692739.

Sender transmits
10 000006 000007 000000 000000 000000 692739.

Receiver checks authenticator.
Easy to prove upper bound
on success chance of forgery.

But the success chance
is unacceptably high!
"Provable weak security."

Replace 6 digits, $p = 1000003$
with 128 bits, $p = 2^{130} - 5$;
one 128-bit multiplication
for each 128-bit message chunk.
Then success chance of forgery
is small enough to be ignored.
"Provable strong security."

Fewer multiplications

Provably secure authenticators $(m_1 r_1 + m_2 r_2 + \cdots) + s_n$: 1974 Gilbert/MacWilliams/Sloane.

Crypto 1999, Black/Halevi/ Krawczyk/Krovetz/Rogaway (crediting Carter/Wegman): Replace $m_1 r_1 + m_2 r_2$ with $(m_1 + r_1)(m_2 + r_2)$, replace $m_3 r_3 + m_4 r_4$ with $(m_3 + r_3)(m_4 + r_4)$, etc. Half as many multiplications!

Same speedup idea as 1968 Winograd matrix mult.

Fewer secret r's

FOCS 1979, Wegman/Carter:
Another authentication function;
fewer secrets $r_1, r_2, \ldots$.

1987 Karp/Rabin, 1981 Rabin:
Another authentication function;
extremely short secret r,
but expensive to generate.

1993 den Boer; independently
1994 Taylor; independently 1994
Johansson/Kabatianskii/Smeets:
Another authentication function;
extremely short secret r,
trivial to generate.

den Boer et al. authenticator:

$m_1 r^5 + m_2 r^4 + \cdots + m_5 r + s_n$.

Oops, lost the 2$\times$ speedup!

2007 Bernstein, using
1970 Winograd speedup idea:
Another authentication function;
extremely short secret r,
trivial to generate;
half as many multiplications.
cr.yp.to/papers.html#pema

$(((r + m_1)(r^2 + m_2) + m_3)$
$\cdot (r^4 + m_4) + m_5)r + s_n$ etc.

Lower-level speedups

Typically gain another
factor of 2 or more
from fast field arithmetic.

Many choices. Which prime?
Or non-prime finite field?
How to encode messages?
How to split integers?
How to build arithmetic
from CPU instructions?

With careful choices, can
compute secure authenticator
on common CPUs in just
a few cycles per message byte.

Should r be reused?

Secrets $r, s_1, s_2, s_3, \ldots$
(1979 Wegman/Carter):
minimum length but
each message accesses
two segments of array.

Secrets $r_1, s_1, r_2, s_2, r_3, s_3, \ldots$
(2006 Lange):
each message accesses
only one segment of array.
If receiver enforces
non-reuse of nonces
then this structure
also stops "re-forgeries."

3. Ciphers

2.: Sender generates independent uniform random secrets $r_1, s_1, \ldots$.
Shares with receiver.
Computes authenticators.

2. + **3**.: Sender generates uniform random secret 128-bit string k.
Shares with receiver.
Computes $(r_1, s_1, \ldots) = (\text{AES}_k(0), \text{AES}_k(1), \text{AES}_k(2), \ldots)$, in advance or upon demand.
Computes authenticators.

Advantage of this change:
Much shorter secret key;
much less expensive
to generate and share.

Disadvantage of this change:
Can't prove security.
New $r_1, s_1, \ldots$ are not
independent uniform random.

Standard security *conjecture*:
$(\mathrm{AES}_k(0), \mathrm{AES}_k(1), \mathrm{AES}_k(2), \ldots)$
is very hard to distinguish
from a uniform random string.
Conjecture seems reasonable.

Each message: Authentication
uses 32 bytes from AES.
2 blocks; $\geq$ 300 CPU cycles.
Huge cost for short messages.

(Plus extra costs: key expansion;
protection against timing leaks;
more AES blocks if encrypting.)

Many faster alternatives.
See, e.g., my Salsa20 cipher
and other ciphers in 3rd round of
ECRYPT Stream Cipher Project.
Salsa20/8 generates 64 bytes
in 128 Core 2 cycles.

4. Diffie-Hellman functions

2. + **3**.: Alice generates k.
Alice shares k with Bob
through a secret authentic
communications channel.
Use k to authenticate
messages on other channels.

2. + **3**. + **4**.: Alice, Bob use
an authentic *non-secret*
communications channel
to agree on a secret k.
Use k to authenticate
messages on other channels.

1976 Diffie/Hellman:

Standardize $q = 2^{262} - 5081$.

$$
\begin{tikzcd}
\begin{array}{c}\text{Alice's}\\ \text{secret key } a\end{array} \arrow[d] \arrow[dd, bend right] & & \begin{array}{c}\text{Bob's}\\ \text{secret key } b\end{array} \arrow[d] \arrow[dd, bend left] \\
\begin{array}{c}\text{Alice's}\\ \text{public key}\\ 4^a \bmod q\end{array} \arrow[drr] & & \begin{array}{c}\text{Bob's}\\ \text{public key}\\ 4^b \bmod q\end{array} \arrow[dll] \\
\begin{array}{c}\{\text{Alice}, \text{Bob}\}\text{'s}\\ \text{shared secret}\\ 4^{ab} \bmod q\end{array} & = & \begin{array}{c}\{\text{Bob}, \text{Alice}\}\text{'s}\\ \text{shared secret}\\ 4^{ab} \bmod q\end{array}
\end{tikzcd}
$$

Compute hash k of $4^{ab} \bmod q$.

Bad news: Attacker can find a and b by "index calculus."

To protect against this attack, replace $2^{262} - 5081$ with a much larger prime.
Much slower arithmetic.

Alternative:
Elliptic-curve cryptography.
Replace $\{1, 2, \ldots, 2^{262} - 5082\}$ with a comparable-size "safe elliptic-curve group."
Somewhat slower arithmetic.

Recent ECC speed news

1. New DH speed records
using “Curve25519” curve:
958000 Pentium 4 cycles;
641000 Pentium M cycles.
(PKC 2006, Bernstein)

Same curve, 64-bit CPUs:
386000 Core 2 cycles;
307000 Opteron cycles.
(SPEED 2007, Gaudry/Thomé)

See eBATS (ECRYPT
Benchmarking of
Asymmetric Systems):
`www.ecrypt.eu.org/ebats`

2. Special hyperelliptic curves should achieve better speeds. See ECC 2006 Bernstein/Lange survey "Elliptic vs. hyperelliptic."

But need serious computation to find secure special curves.

3. New curve shape
(2007 Edwards)
leads to new speed records
(Asiacrypt 2007 et al.,
Bernstein/Lange)
for elliptic-curve computations.
"Elliptic strikes back."

What are Edwards curves?

cr.yp.to/newelliptic.html

Example: Define $q = 2^{255} - 19$
and $d = 1 - 1/121666$.
Then the Edwards curve
$x^2 + y^2 = 1 + dx^2y^2$ over $\mathbf{F}_q$
is equivalent to Curve25519.

The Edwards addition law
$(x_1, y_1) + (x_2, y_2) =$

$$\left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2}\right)$$

works for *all* pairs
of points on this curve.
Denominators are never 0.

With coordinates $(X : Y : Z)$ representing Edwards $(X/Z, Y/Z)$, can add using $10\mathbf{M} + 1\mathbf{S} + 1\mathbf{D}$.

Check for special cases? Not required, but saves time. Can double using $3\mathbf{M} + 4\mathbf{S}$.

Consistently fewer mults than, e.g., Jacobian coordinates. Fewer mults than Montgomery for large scalars.

Implementation in progress. Expect new speed records for Curve25519 etc.

5. Public-key signature systems

Summary of costs for sender
to authenticate B blocks
in M messages to R receivers:
1 public-key generation;
R shared-secret generations;
M cipher invocations;
B multiplications.
Similar costs for receiver.

Alternative: Public-key signatures.
1 public-key generation;
S signature generations
for S *unique* messages;
M verifications for receivers.

State-of-the-art signatures

Standardize hash function H;
Q, order p, on Curve25519.

Signer has 32-byte secret key $n \in \{0, 1, \ldots, 2^{256} - 1\}$; 32-byte public key, compressed $K = nQ$.

To verify $(m, \text{compressed } R, t)$:
verify $tQ = H(R, m)R + K$.

To sign m: generate a secret s;
$R = sQ$; $t = H(R, m)s + n \bmod p$.

(first similar idea: 1985 ElGamal;
many generalizations, variations;
these choices: 2006 van Duin)

To verify a batch

$t_1 Q - h_1 R_1 = K_1$,

$t_2 Q - h_2 R_2 = K_2$,

$\vdots$,

$t_{100} Q - h_{100} R_{100} = K_{100}$:

Verify linear combination

$(v_1 t_1 + \cdots + v_{100} t_{100}) Q$

$- v_1 h_1 R_1 - \cdots - v_{100} h_{100} R_{100}$

$- v_1 K_1 - \cdots - v_{100} K_{100} = 0$

for random 128-bit $v_1, \ldots, v_{100}$.

(Eurocrypt 1994, Naccache et al.; Eurocrypt 1998, Bellare et al.)

Use subtractive multi-scalar multiplication algorithm:
if $n_1 \geq n_2 \geq \cdots$ then
$n_1 P_1 + n_2 P_2 + n_3 P_3 + \cdots =$
$(n_1 - q n_2) P_1 + n_2 (q P_1 + P_2) +$
$n_3 P_3 + \cdots$ where $q = \lfloor n_1 / n_2 \rfloor$.
(Eurocrypt 1994, de Rooij, credited to Bos and Coster; see also tweaks by Wei Dai, 2007)

Only ≈ 25.2 curve adds/bit to verify 100 signatures.

Can use Jacobian coordinates, but Edwards is much faster!