Pseudo-random # generators

Why? eg. secret keys for public key crypto

Want: - deterministic alg. to generate a stream of bits from a seed.

- internal state (can't access)
- can see bit stream
- property ①: can't guess next output
- property ②: statistically random (no correlations, 50-50 bits etc.)
- in practice, it is periodic but with a huge (infeasible) period.

Examples. 60's 70's mainframes

RANDU $s_{j+1} = 65539 \cdot s_j \mod 2^{31}$

$s_0 =$ seed

reveal a few bits of s_j

$s_0 \longrightarrow s_0 \cdot \alpha \longrightarrow s_0 \alpha^2 \longrightarrow s_0 \alpha^3 \longrightarrow \cdots$

BUT graph $(s_{3k}, s_{3k+1}, s_{3k+2})$ in 3D, see patterns

→ not very random

≠ truly horrible" — Donald Knuth.

Linear congruential generators

$$s_n = a\, s_{n-1} + b \pmod m$$

look random for good a, b — fast
— not cryptographically secure

Cryptographic PRNG's today:

① many hashing-based

$$s_{j+1} = h(s_j)$$

SSL uses MD5 hash (?)

② Blum-Blum-Shub

$n = pq$ $\quad p, q \equiv 3 \pmod 4$

$$s_j = s_{j-1}^2 \pmod n$$

output a few least sign. bits

It is possible to prove you can't predict bits w/o factoring n.

Dual_EC_DRBG

"deterministic random bit generator"

"elliptic curve"

p prime
E elliptic curve mod p
P, Q on $E(\mathbb{F}_p)$

internal $s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \cdots$ $\quad s_{j+1} = x(s_j P)$

output $\quad \text{trunc}(x(s_1 Q))$

drop the least sig 16 bits

Security Issue!

- truncation drops very few bits
  → could guess $x(s_i Q)$.
- then know $s_i Q$.

$\Rightarrow \quad m \cdot s_i Q = s_i \, m Q = s_i (P)$ if $P = mQ$.

2000-2004 ish
NIST standardized

2007
researchers @ Microsoft pointed out an issue

2013 – Snowden leaks "Bullrun"

2014 – removed as a standard.

- Conclusion: if we know m s.t. $P = mQ$
  then we can recover internal state.

- Fear: NIST (NSA) may have a backdoor. (choose Q, m, compute P)

Fix (what should NIST have done?)

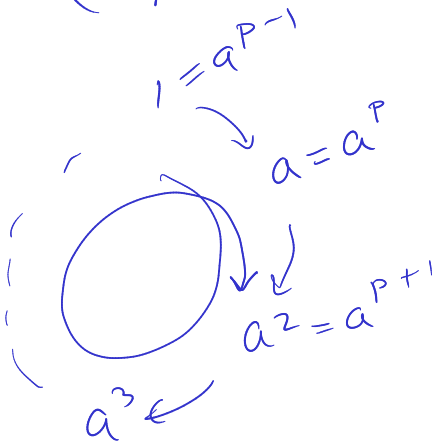
"Nothing up my sleeve" numbers

choose P, Q w/ say, x-coords based on digits of π.

$\Rightarrow$ ECDLP rules out knowing m.

Finding square roots mod p.

$(\mathbb{Z}/p\mathbb{Z})^* = p-1$



Let $a \in (\mathbb{Z}/p\mathbb{Z})^*$

Goal: find $\sqrt{a}$?

If $p \equiv 3 \pmod 4$ then $p+1 \equiv 0 \pmod 4$

So $a^{\frac{p+1}{4}}$ is well-defined.

and $\left(a^{\frac{p+1}{4}}\right)^4 = a^{p+1} = a^2 \pmod p$

i.e. $\left(a^{\frac{p+1}{4}}\right)^2 = \pm a \pmod p$ → only 2 square roots

Claim: a and $-a$ can't both be squares.

Pf. If a and $-a$ are both squares, say $t^2 = -a$, $s^2 = a$,

then $(t/s)^2 = t^2/s^2 = -a/a = -1$.

But $-1 = g^{\frac{p-1}{2}}$ for generator g

then $2 \cdot L_g(t/s) = L_g(-1) = \frac{p-1}{2}$

BUT $\frac{p-1}{2}$ is odd if $p \equiv 3 \ (4)$. ☐

Algorithm: (to find $\sqrt{a} \bmod p \equiv 3 \ (4)$)

① Compute $k = a^{\frac{p+1}{4}}$.

② Case I: $k^2 = a$ we are done.

Case II: $k^2 = -a$ we conclude a is not a square. (no square root).

Tonelli-Shanks algorithm in general.