# Verifiably random secure curves

Daniel J. Bernstein
Tung Chou
Chitchanok Chuengsatiansup
Andreas Hülsing
Tanja Lange
Ruben Niederhagen
Christine van Vredendaal

May 13, 2014

# NSA/NIST FUD

The NIST elliptic curves are behind the state of the art:

- Chosen by Jerry Solinas at **NSA**.

# NSA/NIST FUD

The NIST elliptic curves are behind the state of the art:

- Chosen by Jerry Solinas at **NSA**.
- Coefficients produced from NSA's **SHA-1**.

# NSA/NIST FUD

The NIST elliptic curves are behind the state of the art:

- Chosen by Jerry Solinas at **NSA**.
- Coefficients produced from NSA's **SHA-1**.
- NIST P-224 is **not twist-secure**.
- etc.

Let's make some new curves.

# Verifiable randomness

Produce **verifiably random** numbers
using a **secure hash** so that nobody has to trust us.

- 2000: Certicom Research "Standards for Efficient Cryptography 2: Recommended Elliptic Curve Domain Parameters", Version 1.0.
- 2000: IEEE Std 1363-2000 "IEEE Standard Specifications for Public-Key Cryptography".
- 2001: ANSI X9.63 "Public Key Cryptography For The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography".
- 2010: Certicom Research (Daniel R. L. Brown) "Standards for Efficient Cryptography 2: Recommended Elliptic Curve Domain Parameters", Version 2.0.

# On the importance of verifiable randomness

2014.01.13 Daniel R. L. Brown:

`1. Pseudorandomness protects effectively (as possible for ECC) against the spectral weakness necessary to hypothesize a malicious NIST P256.`

`2. Rigidity protects against the spectral weakness only by invoking assumptions about spectral weakness (*).`

`3. Protecting against attacks, such as the hypothetical spectral weakness, is more important than (subsumes?) protecting against malicious generation.`

# On the importance of verifiable randomness

2014.01.13 Daniel R. L. Brown:

```
1.  Pseudorandomness protects effectively (as
possible for ECC) against the spectral weakness
necessary to hypothesize a malicious NIST P256.

2.  Rigidity protects against the spectral weakness
only by invoking assumptions about spectral weakness
(*).

3.  Protecting against attacks, such as the
hypothetical spectral weakness, is more important
than (subsumes?)  protecting against malicious
generation.
```

Does anyone here know what "**spectral weakness**" means?

Picture credit: eyerayofthebeholder.blogspot.dk/2014/01/a-story-driven-weakness-for-allip.html

# Freshly made from the best ingredients

Take the NIST P-256 prime $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

# Freshly made from the best ingredients

Take the NIST P-256 prime $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

Generate random seeds $s$ and hashes $B = H(s)$.
Hash function $H$:
**Keccak** with 256-bit output (i.e., `keccakc512`).

# Freshly made from the best ingredients

Take the NIST P-256 prime $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

Generate random seeds $s$ and hashes $B = H(s)$.
Hash function $H$:
**Keccak** with 256-bit output (i.e., `keccakc512`).

If the elliptic curve $x^3 - 3x + B \bmod p$
does not meet standard security criteria **plus twist-security**,
start over. (This happens tens of thousands of times!)

# Freshly made from the best ingredients

Take the NIST P-256 prime $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

Generate random seeds $s$ and hashes $B = H(s)$.
Hash function $H$:
**Keccak** with 256-bit output (i.e., `keccakc512`).

If the elliptic curve $x^3 - 3x + B \bmod p$
does not meet standard security criteria **plus twist-security**,
start over. (This happens tens of thousands of times!)

Same with NIST P-224 prime $2^{224} - 2^{96} + 1$.

# Freshly made from the best ingredients

Take the NIST P-256 prime $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

Generate random seeds $s$ and hashes $B = H(s)$.
Hash function $H$:
**Keccak** with 256-bit output (i.e., `keccakc512`).

If the elliptic curve $x^3 - 3x + B \bmod p$
does not meet standard security criteria **plus twist-security**,
start over. (This happens tens of thousands of times!)

Same with NIST P-224 prime $2^{224} - 2^{96} + 1$.

Also with NIST P-384 prime $2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$.
`keccakc512` is too small here so we switched to `keccakc768`.

# Random seeds for your verification pleasure

224: `3CC520E9434349DF680A8F4BCADDA648`
`D693B2907B216EE55CB4853DB68F9165`

256: `3ADCC48E36F1D1926701417F101A75F0`
`00118A739D4686E77278325A825AA3C6`

384: `CA9EBD338A9EE0E6862FD329062ABC06`
`A793575A1C744F0EC24503A525F5D06E`

# The $B$ values in $x^3 - 3x + B$

224: `BADA55ECFD9CA54C0738B8A6FB8CF4CC`
`F84E916D83D6DA1B78B622351E11AB4E`

256: `BADA55ECD8BBEAD3ADD6C534F92197DE`
`B47FCEB9BE7E0E702A8D1DD56B5D0B0C`

384: `BADA55EC3BE2AD1F9EEEA5881ECF95BB`
`F3AC392526F01D4CD13E684C63A17CC4`
`D5F271642AD83899113817A61006413D`

# The $B$ values in $x^3 - 3x + B$

224: **BADA55EC**FD9CA54C0738B8A6FB8CF4CC
F84E916D83D6DA1B78B622351E11AB4E

256: **BADA55EC**D8BBEAD3ADD6C534F92197DE
B47FCEB9BE7E0E702A8D1DD56B5D0B0C

384: **BADA55EC**3BE2AD1F9EEEA5881ECF95BB
F3AC392526F01D4CD13E684C63A17CC4
D5F271642AD83899113817A61006413D

1999 Michael Scott "Re: NIST annouces set of Elliptic Curves":

```
Consider now the possibility that one in a million of all
curves have an exploitable structure that "they" know
about, but we don't..  Then "they" simply generate a
million random seeds until they find one that generates
one of "their" curves.  Then they get us to use them.  And
remember the standard paranoia assumptions apply - "they"
have computing power way beyond what we can muster.  So
maybe that could be 1 billion.

A much simpler approach would generate more trust.  Simply
select B as an integer formed from the maximum number of
digits of pi that provide a number B which is less that
p.Then keep incrementing B until the number of points on
the curve is prime.  Such a curve will be accepted as
"random" as all would accept that the decimal digits of pi
have no unfortunate interaction with elliptic curves.  We
would all accept that such a curve had not been specially
"cooked".

So, sigh, why didn't they do it that way?  Do they want to
be distrusted?
```

# Brainpool to the rescue

2005 "ECC Brainpool standard curves and curve generation" generates deterministic seeds from $\pi$ and $e$.

`brainpoolP256r1`:

```
p: A9FB57DBA1EEA9BC3E660A909D838D72
   6E3BF623D52620282013481D1F6E5377
A: 7D5A0975FC2C3057EEF67530417AFFE7
   FB8055C126DC5C6CE94A4B44F330B5D9
B: 26DC5C6CE94A4B44F330B5D9BBD77CBF
   958416295CF7E1CE6BCCDC18FF8C07B6
```

# Brainpool to the rescue (or maybe not)

2005 "ECC Brainpool standard curves and curve generation" generates deterministic seeds from $\pi$ and $e$.

`brainpoolP256r1`:

```
p: A9FB57DBA1EEA9BC3E660A909D838D72
   6E3BF623D52620282013481D1F6E5377
A: 7D5A0975FC2C3057EEF67530417AFFE7
   FB8055C126DC5C6CE94A4B44F330B5D9
B: 26DC5C6CE94A4B44F330B5D9BBD77CBF
   958416295CF7E1CE6BCCDC18FF8C07B6
```

# Brainpool to the rescue (or maybe not)

2005 "ECC Brainpool standard curves and curve generation" generates deterministic seeds from $\pi$ and $e$.

```
brainpoolP256r1:
p: A9FB57DBA1EEA9BC3E660A909D838D72
   6E3BF623D52620282013481D1F6E5377
A: 7D5A0975FC2C3057EEF67530417AFFE7
   FB8055C126DC5C6CE94A4B44F330B5D9
B: 26DC5C6CE94A4B44F330B5D9BBD77CBF
   958416295CF7E1CE6BCCDC18FF8C07B6
```

Screwed up data flow in hash inputs; still uses SHA-1; not twist-secure.

Let's make an **NSA-free** replacement with **sensible data flow**.

And let's stick to the NIST primes.

# Nothing up our sleeves

Constants already used: $\sin 1$; $\pi/4 = \arctan 1$; $e = \exp 1$.
Start from $\cos 1$.

# Nothing up our sleeves

Constants already used: $\sin 1$; $\pi/4 = \arctan 1$; $e = \exp 1$.
Start from $\cos 1$.

Generate the full 160-bit seed
as 32-bit counter followed by $\cos 1$.

(16-bit counter would have been unsafe:
more than $1/1000$ chance of failing to find secure curve.)

# Nothing up our sleeves

Constants already used: $\sin 1$; $\pi/4 = \arctan 1$; $e = \exp 1$.
Start from $\cos 1$.

Generate the full 160-bit seed
as 32-bit counter followed by $\cos 1$.

(16-bit counter would have been unsafe:
more than $1/1000$ chance of failing to find secure curve.)

To avoid the Brainpool problems:

- Don't concatenate SHA-1 outputs.
  Use maximum-security full-length SHA-3-512.
- Generate $B$ seed as complement of $A$ seed.
  Guaranteed to be different.

Sage computer-algebra system computing 128 bits of $\cos 1$:

```
sage -c 'print RealField(128)(cos(1)).str(16)[2:34]'
8a51407da8345c91c2466d976871bd2a
```

We started computations recently for the NIST P-224 prime and already found a secure twist-secure curve from seed `000000B8 8A51407DA8345C91C2466D976871BD2A`.

Here are $A, B$ (please verify with your own SHA-3 software):

```
7144BA12CE8A0C3BEFA053EDBADA555A
42391FC64F052376E041C7D4AF23195E
BD8D83625321D452E8A0C3BB0A048A26
115704E45DCEB346A9F4BD9741D14D49,
```

```
5C32EC7FC48CE1802D9B70DBC3FA574E
AF015FCE4E99B43EBE3468D6EFB2276B
A3669AFF6FFC0F4C6AE4AE2E5D74C3C0
AF97DCE17147688DDA89E734B56944A2
```

Sage computer-algebra system computing 128 bits of $\cos 1$:

```
sage -c 'print RealField(128)(cos(1)).str(16)[2:34]'
8a51407da8345c91c2466d976871bd2a
```

We started computations recently for the NIST P-224 prime and already found a secure twist-secure curve from seed `000000B8 8A51407DA8345C91C2466D976871BD2A`.

Here are $A, B$ (please verify with your own SHA-3 software):

```
7144BA12CE8A0C3BEFA053EDBADA555A
42391FC64F052376E041C7D4AF23195E
BD8D83625321D452E8A0C3BB0A048A26
115704E45DCEB346A9F4BD9741D14D49,
```

```
5C32EC7FC48CE1802D9B70DBC3FA574E
AF015FCE4E99B43EBE3468D6EFB2276B
A3669AFF6FFC0F4C6AE4AE2E5D74C3C0
AF97DCE17147688DDA89E734B56944A2
```

# Lessons and credits

"Verifiably random" curves,
even with "deterministic" seeds,
do not stop the attacker
from generating a curve
with a one-in-a-million weakness.

safecurves.cr.yp.to/bada55.html

Computation credits:
Saber cluster at Technische Universiteit Eindhoven;
ISF K10 cluster at University of Haifa.