Applied cryptography

How to use 'secure' crypto

Andreas Hülsing

11 May 2015

Motivation

Consider someone wants to securely exchange a secret key and asks you which 'Crypto' to use.

What would you answer? And why?

Motivation

Consider someone wants to securely exchange a secret key and asks you which 'Crypto' to use.

What would you answer? And why?

Motivation II

Many aspects:

- Which constructions?
- Which parameters?
- Which implementation?

Motivation II

Many aspects:

- Which constructions?
- Which parameters?
- Which implementation?

Motivation II

Many aspects:

- Which constructions?
- Which parameters?
- Which implementation?

Todays goals

- Which constructions?
- How to select secure parameters.
- (How to select a secure implementation.)

Example

ECDH:

The literature models ephemeral ECDH as the following protocol $\mathrm{ECDH}_{E,P}$, Diffie–Hellman key exchange using a point P on an elliptic curve E:

1. Alice generates private int a and sends the ath multiple of P on E.
2. Bob generates a private int b and sends bP.
3. Alice computes abP as the ath multiple of bP.
4. Bob computes abP as the bth multiple of aP.
5. Alice and Bob derived secret key from abP.

- Scheme is provably secure.
- Security Parameter: Size p of the underlying (prime-order) field $\mathbb{F}_p$.
- Assumed security: $\approx 0.886\sqrt{p}$ (Pollard's rho method).

Example

ECDH:

The literature models ephemeral ECDH as the following protocol $\mathrm{ECDH}_{E,P}$, Diffie–Hellman key exchange using a point P on an elliptic curve E:

1. Alice generates private int a and sends the ath multiple of P on E.
2. Bob generates a private int b and sends bP.
3. Alice computes abP as the ath multiple of bP.
4. Bob computes abP as the bth multiple of aP.
5. Alice and Bob derived secret key from abP.

- Scheme is provably secure.
- Security Parameter: Size p of the underlying (prime-order) field $\mathbb{F}_p$.
- Assumed security: $\approx 0.886\sqrt{p}$ (Pollard's rho method).

Assumed security

- What does $\approx 0.886\sqrt{p}$ *s*ecure mean?
- What does x-secure mean in general?
- How long is an x-secure scheme secure?
- Against whom?

Assumed security

- What does $\approx 0.886\sqrt{p}$ *s*ecure mean?
- What does x-secure mean in general?
- How long is an x-secure scheme secure?
- Against whom?

Assumed security

- What does $\approx 0.886\sqrt{p}$ *s*ecure mean?
- What does x-secure mean in general?
- How long is an x-secure scheme secure?
- Against whom?

Assumed security

- What does $\approx 0.886\sqrt{p}$ *s*ecure mean?
- What does x-secure mean in general?
- How long is an x-secure scheme secure?
- Against whom?

Outline

1. **Estimating the security of parameters:**
   The Lenstra – Verheul heuristics.
2. **Why you should check all parameters:**
   Nothing up my sleeves numbers.
3. **Why open-source is more than a philosophical question:**
   Kleptography.

Estimating the security of parameters

How secure is a parameter set?

Lenstra and Verheul (2001): Selecting Cryptographic Key Sizes
Introduces:

- **security level** notion,
- method to measure attack effort
- method to estimate technological progress
- method to estimate cryptanalytic progress
- suggestion for adequate security level

Security level

- Defines what it means for a scheme to be x-secure.
- Attack effort based notion.
- Sometimes called bit security.

Security level II

Definition (Lenstra 2004)

In general, a cryptographic system offers security level λ if a successful generic attack can be expected to require effort approximately $2^{\lambda-1}$.

Example

A secure symmetric key encryption scheme with λ-bit keys is traditionally expected to provide a security level of λ bits as the best generic attack is exhaustive key search.

Security level II

Definition (Lenstra 2004)

In general, a cryptographic system offers security level λ if a successful generic attack can be expected to require effort approximately $2^{\lambda-1}$.

Example

A secure symmetric key encryption scheme with λ-bit keys is traditionally expected to provide a security level of λ bits as the best generic attack is exhaustive key search.

Cost of an attack effort

- Attack effort $\neq$ time.
- Attacks assumed to be fully parallelizable.
- Hence, measure effort in *dollardays*.

$\Rightarrow$ Allows to ignore implementation details.

Example

- Assume attack takes d days on c dollar device.
- Can also be run on wc dollar device in d/w days.
- Attack effort is dc dollardays.

Cost of an attack effort

- Attack effort $\neq$ time.
- Attacks assumed to be fully parallelizable.
- Hence, measure effort in *dollardays*.

$\Rightarrow$ Allows to ignore implementation details.

Example

- Assume attack takes d days on c dollar device.
- Can also be run on wc dollar device in d/w days.
- Attack effort is dc dollardays.

Adequate security level

Which security level do we consider secure?

- Lenstra & Verheul use DES in 1982.
- Extensions possible.
- Was DES secure against NSA in 1982?
- Too strong for your application?

The Data Encryption Standard

- 56-bit block cipher.
- Published 1977 by U.S. Department of Commerce.
- Reaffirmation after 5 years.
- 1982: Widely adopted, no better attack than exhaustive key search.
- Considered 'adequate security for commercial applications' in 1982.

Baseline

56-bit security in 1982 seemed adequate.

Cost of breaking DES

- 1980: 2 days on US$50 million device (estimate).
- $\Rightarrow 100M$ dollardays.
- This estimate used 1980 money and technology!
- 1993: DES key search engine proposed that would require about $150K$ dollardays
- 1998: parallel hardware device built for US130K$ including design overhead, and used to crack the DES in a matter of days.
- Today: Even 3DES considered insecure...

Cost of breaking DES

- 1980: 2 days on US$50 million device (estimate).
- $\Rightarrow$ $100M$ dollardays.
- This estimate used 1980 money and technology!
- 1993: DES key search engine proposed that would require about $150K$ dollardays
- 1998: parallel hardware device built for US130K$ including design overhead, and used to crack the DES in a matter of days.
- Today: Even 3DES considered insecure...

Cost of breaking DES

- 1980: 2 days on US$50 million device (estimate).
- $\Rightarrow 100M$ dollardays.
- This estimate used 1980 money and technology!
- 1993: DES key search engine proposed that would require about $150K$ dollardays
- 1998: parallel hardware device built for US130K$ including design overhead, and used to crack the DES in a matter of days.
- Today: Even 3DES considered insecure...

Cost of breaking DES

- 1980: 2 days on US$50 million device (estimate).
- $\Rightarrow 100M$ dollardays.
- This estimate used 1980 money and technology!
- 1993: DES key search engine proposed that would require about $150K$ dollardays
- 1998: parallel hardware device built for US130K$ including design overhead, and used to crack the DES in a matter of days.
- Today: Even 3DES considered insecure...

Cost of breaking DES

- 1980: 2 days on US$50 million device (estimate).
- $\Rightarrow 100M$ dollardays.
- This estimate used 1980 money and technology!
- 1993: DES key search engine proposed that would require about $150K$ dollardays
- 1998: parallel hardware device built for US130K$ including design overhead, and used to crack the DES in a matter of days.
- Today: Even 3DES considered insecure...

Technological progress

Definition (Moore's Law, adoption in (Lenstra 2004))

The cost of any fixed attack effort drops by a factor 2 every 18 months.

- More likely to remain valid in future.
- Also covers inflation.

Technological progress

Definition (Moore's Law, adoption in (Lenstra 2004))

The cost of any fixed attack effort drops by a factor 2 every 18 months.

- More likely to remain valid in future.
- Also covers inflation.

Applying Moore's Law to DES

- 1980: $100M$ dollardays
- 1982: $40M$ dollardays as $40 \approx 100/2^{24/18}$

Baseline

$40M$ dollardays considered adequate.

- Can be changed to $x40M$ dollardays.
- $x > 1$ if considered to low.
- $0 > x > 1$ if considered overkill.

Applying Moore's Law to DES

- 1980: $100M$ dollardays
- 1982: $40M$ dollardays as $40 \approx 100/2^{24/18}$

Baseline

$40M$ dollardays considered adequate.

- Can be changed to $x40M$ dollardays.
- $x > 1$ if considered to low.
- $0 > x > 1$ if considered overkill.

Applying Moore's Law to DES

- 1980: $100M$ dollardays
- 1982: $40M$ dollardays as $40 \approx 100/2^{24/18}$

Baseline

$40M$ dollardays considered adequate.

- Can be changed to $x40M$ dollardays.
- $x > 1$ if considered to low.
- $0 > x > 1$ if considered overkill.

Estimates for symmetric crypto I

Hash function security

What digest size is secure today?

- Baseline: 56 bit adequate 1982
- x bit adequate in 2015
- $2015 - 1982 = 33$ years
- $22 \cdot 1.5$ years
- $x = 56 + 22 = 78$ bits
- Digest size $n \geq 156$ (remember birthday attacks!)
- Only secure until 2015!

$\Rightarrow$ If symmetric scheme allows for significantly better attacks than generic ones, it is considered insecure.

Estimates for symmetric crypto II

General estimates

For a primitive $\mathcal{S}$ that runs in approx. same time as DES:

- A security level λ provides security until year

$$y(\lambda) = 1982 + \frac{3(\lambda - 56)}{2}$$

- In year y, to be secure one needs at least security level

$$\lambda(y) = 56 + \frac{2(y - 1982)}{3}$$

If $\mathcal{S}$ is $s > 0$ faster than DES, either add $\log_2 s$ to $\lambda(y)$ or subtract $1.5 \log_2 s$ from $y(\lambda)$.

Estimates for asymmetric crypto

- Estimates using dollardays metric.
- If symmetric scheme allows for significantly better attacks than generic ones, it is considered insecure.
  - Works because of variety of available schemes.
  - Does not work in asymmetric world with little choices of assumptions.
- Considers cryptanalytic effort.

Estimates for asymmetric crypto, general procedure

- Take known attack effort on one instance at a given time in dollardays (e.g. RSA: $400M$ dollardays for 1024 bit modulo in 2004)
- Extrapolate to other instances using attack complexities (e.g. RSA: extrapolate to 2048 bit using NFS complexity ≈ 2 billion times costlier)
- Use Moore's law and estimate for cryptanalytic progress to estimate attack effort in year y (RSA: costs assumed to drop by factor of 2 every 9 months)
- Instance considered secure as long as effort $> 40M$ dollardays.

Special case: Discrete log

- Attack complexity $\approx \sqrt{|\langle g \rangle|} \approx \sqrt{p}$ for elliptic curve over prime order field $\mathbb{F}_p$ as of Polard's rho method.
- Cryptanalytic progress stable for several years.
- Fall-back to symmetric method: For security level λ use p with 2λ bits.

Security level vs. main security parameter

- Symmetric encryption: Use λ-bit keys.
- Hash function: Use 2λ-bit digests.
- RSA: See http://keylength.com.
- ECC: Use curves over prime order fields $\mathbb{F}_p$ with 2λ-bit primes.
- Short-cut: See http://keylength.com.

Why you should check all parameters

Two kinds of parameters:

- The obvious ones: 'Security parameters' aka 'key-length'
- The less obvious ones: Scheme defining parameters, especially constants.

Example I

ECDH:

The literature models ephemeral ECDH as the following protocol $\mathrm{ECDH}_{E,P}$, Diffie–Hellman key exchange using a point P on an elliptic curve E:

1. Alice generates private int a and sends the ath multiple of P on E.
2. Bob generates a private int b and sends bP.
3. Alice computes abP as the ath multiple of bP.
4. Bob computes abP as the bth multiple of aP.
5. Alice and Bob derived secret key from abP.

Example II

ECDH in reality

In reality ECDH can be modeled as the following protocol ECDH, between Jerry, Alice and Bob:

1. Jerry generates a curve E, a point P, auxiliary data S with $A(E, P, S) = 1$.
2. Alice and Bob verify that $A(E, P, S) = 1$.
3. Alice generates private int a and sends the ath multiple of P on E.
4. Bob generates a private int b and sends bP.
5. Alice computes abP as the ath multiple of bP.
6. Bob computes abP as the bth multiple of aP.
7. Alice and Bob derived secret key from abP.

Example III

- Someone selects the curve you are using!
- Jerry might be NIST (= NSA?), BSI, ANSSI....
- What if Jerry knows a secret vulnerability?
- There are many insecure curves! (Only 1 out of $> 32,000$ curves fulfills all criteria from http://safecurves.cr.yp.to)
- Let's assume the secret vulnerability applies to 1 out of 2^{20} curves.
- What can Jerry do?

Lets take Jerry's place.

Warm-up:

As a first example, we look at the FRP256V1 standard published in 2011 by the Agence nationale de la sécurité des systèmes d'information (ANSSI). This curve is $y^2 = x^3 - 3x + b$ over $\mathbb{F}_p$, where

$$\begin{aligned} b &= \texttt{0xEE353FCA5428A9300D4ABA754A44C00FD} \\ &\quad \texttt{FEC0C9AE4B1A1803075ED967B7BB73F}, \\ p &= \texttt{0xF1FD178C0B3AD58F10126DE8CE42435B3} \\ &\quad \texttt{961ADBCABC8CA6DE8FCF353D86E9C03}. \end{aligned}$$

- $A(E, P, S) = 1$ constant function.
- Generating a vulnerable curve with this $A(E, P, S)$ is trivial.

Warm-up:

As a first example, we look at the FRP256V1 standard published in 2011 by the Agence nationale de la sécurité des systèmes d'information (ANSSI). This curve is $y^2 = x^3 - 3x + b$ over $\mathbb{F}_p$, where

$$\begin{aligned} b &= \texttt{0xEE353FCA5428A9300D4ABA754A44C00FD} \\ &\quad \texttt{FEC0C9AE4B1A1803075ED967B7BB73F}, \\ p &= \texttt{0xF1FD178C0B3AD58F10126DE8CE42435B3} \\ &\quad \texttt{961ADBCABC8CA6DE8FCF353D86E9C03}. \end{aligned}$$

- $A(E, P, S) = 1$ constant function.
- Generating a vulnerable curve with this $A(E, P, S)$ is trivial.

NIST's Verifiably Random curve generation

Each NIST P-curve is of the form $y^2 = x^3 - 3x + b$ over a prime field $\mathbb{F}_p$ and is published with a seed s.
The bit length of p is denoted by m.

1. Compute $h_i \leftarrow \text{SHA-1}(s_i)$ for $0 \leq i \leq v$, where $s_i \leftarrow (s + i) \bmod 2^{160}$ and $v = \lfloor (m-1)/160 \rfloor$.
2. Let c be the rightmost $m - 1$ bits of $h_0 || h_1 || \cdots || h_v$. If $c = 0$ or $4c + 27 = 0$, restart with new s.
3. Choose $a, b \in \mathbf{F}_p$ such that $b^2 c \equiv a^3 \pmod{p}$.
4. Check that the curve give by $y^2 = x^3 + ax + b$ over $\mathbf{F}_p$ has suitable order. If not, restart with new s.

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.

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.

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.

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.

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`.

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`.

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`.

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`.

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: BADA55ECFD9CA54C0738B8A6FB8CF4CC
F84E916D83D6DA1B78B622351E11AB4E

256: BADA55ECD8BBEAD3ADD6C534F92197DE
B47FCEB9BE7E0E702A8D1DD56B5D0B0C

384: BADA55EC3BE2AD1F9EEEA5881ECF95BB
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 (Verifiably Pseudorandom)

Simplified Brainpool: Curve is $y^2 = x^3 - ax + b$ over $\mathbb{F}_p$.

1. Obtain a 160-bit seed s_0 from Euler's number.
2. Compute $h_i \leftarrow \text{SHA-1}(s_i)$ for $0 \leq i \leq v$, where $s_i \leftarrow (s + i) \bmod 2^{160}$ and $v = \lfloor (m-1)/160 \rfloor$.
3. Let a be the rightmost $m - 1$ bits of $h_0 || h_1 || \cdots || h_v$. Do some checks. Set $\text{seed}_a \leftarrow s_0$.
4. Set $s_0 \leftarrow (s_0 + i) \bmod 2^{160}$.
5. Compute $h_i \leftarrow \text{SHA-1}(s_i)$ for $0 \leq i \leq v$, where $s_i \leftarrow (s + i) \bmod 2^{160}$ and $v = \lfloor (m-1)/160 \rfloor$.
6. Let b be the rightmost $m - 1$ bits of $h_0 || h_1 || \cdots || h_v$. Do some checks. Set $\text{seed}_b \leftarrow s_0$.
7. Return curve parameters a, b, and seeds seed_a, seed_b for verification.

Brainpool to the rescue

2005 “ECC Brainpool standard curves and curve generation” generates deterministic seeds from π 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.

Brainpool to the rescue (or maybe not)

2005 “ECC Brainpool standard curves and curve generation” generates deterministic seeds from π 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.

Brainpool to the rescue (or maybe not)

2005 "ECC Brainpool standard curves and curve generation"
generates deterministic seeds from π 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 $\cos 1$.

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.

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.

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 did computations for the NIST P-224 prime
and 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 did computations for the NIST P-224 prime
and 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.

Possible Countermeasures

- Generate your own curves?
- Choose parameters based on performance considerations (Curve-25519, NUMS).
- Let many different parties contribute to parameter selection.

Note: this is not only a ECC problem! (See 'Malicious Hashing: Eve's Variant of SHA-1', SAC 2014)

Possible Countermeasures

- Generate your own curves?
- Choose parameters based on performance considerations (Curve-25519, NUMS).
- Let many different parties contribute to parameter selection.

Note: this is not only a ECC problem! (See 'Malicious Hashing: Eve's Variant of SHA-1', SAC 2014)

Possible Countermeasures

- Generate your own curves?
- Choose parameters based on performance considerations (Curve-25519, NUMS).
- Let many different parties contribute to parameter selection.

Note: this is not only a ECC problem! (See 'Malicious Hashing: Eve's Variant of SHA-1', SAC 2014)

Possible Countermeasures

- Generate your own curves?
- Choose parameters based on performance considerations (Curve-25519, NUMS).
- Let many different parties contribute to parameter selection.

Note: this is not only a ECC problem! (See 'Malicious Hashing: Eve's Variant of SHA-1', SAC 2014)

Possible Countermeasures

- Generate your own curves?
- Choose parameters based on performance considerations (Curve-25519, NUMS).
- Let many different parties contribute to parameter selection.

Note: this is not only a ECC problem! (See 'Malicious Hashing: Eve's Variant of SHA-1', SAC 2014)