The discrete-logarithm problem

Define $p = 1000003$.

Easy to prove: $p$ is prime.

Can we find an integer $n \in \{1, 2, 3, \ldots, p - 1\}$ such that $5^n \mod p = 262682$?

Easy to prove: $n \mapsto 5^n \mod p$ permutes $\{1, 2, 3, \ldots, p - 1\}$.

So there exists an $n$ such that $5^n \mod p = 262682$.

Could find $n$ by brute force.

Is there a faster way?
The discrete-logarithm problem
Define $p = 1000003$.
Easy to prove: $p$ is prime.
Can we find an integer $n \in \{1, 2, 3, \ldots, p - 1\}$ such that $5^n \mod p = 262682$?

Easy to prove: $n \mapsto 5^n \mod p$ permutes $\{1, 2, 3, \ldots, p - 1\}$.
So there exists an $n$
such that $5^n \mod p = 262682$.

Could find $n$ by brute force.
Is there a faster way?
The discrete-logarithm problem

Define \( p = 1000003 \).
Easy to prove: \( p \) is prime.

Can we find an integer \( n \in \{1, 2, 3, \ldots, p - 1\} \)
such that \( 5^n \mod p = 262682 \)?

Easy to prove: \( n \mapsto 5^n \mod p \)
permutes \( \{1, 2, 3, \ldots, p - 1\} \).
So there exists an \( n \)
such that \( 5^n \mod p = 262682 \).

Could find \( n \) by brute force.
Is there a faster way?

Typical cryptanalytic application:

Imagine standard \( p = 1000003 \)
in the Diffie-Hellman protocol.

User chooses secret key \( n \),
publishes \( 5^n \mod p = 262682 \).

Can attacker quickly solve the discrete-logarithm problem?
Given public key \( 5^n \mod p \),
quickly find secret key \( n \)?

(Warning: This is one way
to attack the protocol.
Maybe there are better ways.)
The discrete-logarithm problem

Define $p = 1000003$.

Easy to prove: $p$ is prime.

Can we find an integer $n \in \{1, 2, 3, \ldots, p - 1\}$ such that $5^n \mod p = 262682$?

Easy to prove: $n \mapsto 5^n \mod p$ permutes $\{1, 2, 3, \ldots, p - 1\}$.

So there exists an $n$ such that $5^n \mod p = 262682$.

Could find $n$ by brute force.

Is there a faster way?

Typical cryptanalytic application:

Imagine standard $p = 1000003$ in the Diffie-Hellman protocol.

User chooses secret key $n$, publishes $5^n \mod p = 262682$.

Can attacker quickly solve the discrete-logarithm problem?

Given public key $5^n \mod p$, quickly find secret key $n$?

(Warning: This is one way to attack the protocol. Maybe there are better ways.)
The discrete-logarithm problem

Define \( p = 1000003 \).
Easy to prove: \( p \) is prime.

Can we find an integer
\( n \in \{1, 2, 3, \ldots, p - 1\} \)
such that \( 5^n \mod p = 262682 \)?

Easy to prove: \( n \mapsto 5^n \mod p \)
permutes \( \{1, 2, 3, \ldots, p - 1\} \).
So there exists an \( n \)
such that \( 5^n \mod p = 262682 \).
Could find \( n \) by brute force.
Is there a faster way?

Typical cryptanalytic application:

Imagine standard \( p = 1000003 \)
in the Diffie-Hellman protocol.

User chooses secret key \( n \),
publishes \( 5^n \mod p = 262682 \).

Can attacker quickly solve
the discrete-logarithm problem?
Given public key \( 5^n \mod p \),
quickly find secret key \( n \)?

(Warning: This is one way
to attack the protocol.
Maybe there are better ways.)
The discrete-logarithm problem

\[ p = 1000003. \]

Easy to prove: \( p \) is prime.

Can we find an integer \( n \in \{2, 3, \ldots, p - 1\} \) such that \( 5^n \mod p = 262682? \)

Easy to prove: \( n \mapsto 5^n \mod p \) permutes \( \{1, 2, 3, \ldots, p - 1\} \).

So there exists an \( n \) such that \( 5^n \mod p = 262682. \)

Could find \( n \) by brute force.

Is there a faster way?

Typical cryptanalytic application:

Imagine standard \( p = 1000003 \) in the Diffie-Hellman protocol.

User chooses secret key \( n \), publishes \( 5^n \mod p = 262682. \)

Can attacker quickly solve the discrete-logarithm problem?

Given public key \( 5^n \mod p \), quickly find secret key \( n \)?

(Warning: This is one way to attack the protocol.
Maybe there are better ways.)

Relations to ECC:

1. Some DL techniques also apply to elliptic-curve DL problems.
   Use in evaluating security of elliptic curves.

2. Some techniques don't apply.
   Use in evaluating advantages of elliptic curves compared to multiplication.

3. Tricky: Some techniques have extra applications to some curves.
   See Tanja Lange's talk on Weil descent etc.
The discrete-logarithm problem

Define \( p = 1000003 \).

Easy to prove: \( p \) is prime.

Can we find an integer \( n \in \{1, 2, 3, \ldots, p - 1\} \) such that \( 5^n \mod p = 262682 \)?

Easy to prove: \( n \) permutes \( \{1, 2, 3, \ldots, p - 1\} \).

So there exists an \( n \) such that \( 5^n \mod p = 262682 \).

Could find \( n \) by brute force.

Is there a faster way?

Typical cryptanalytic application:

Imagine standard \( p = 1000003 \) in the Diffie-Hellman protocol.

User chooses secret key \( n \),
publishes \( 5^n \mod p = 262682 \).

Can attacker quickly solve the discrete-logarithm problem?

Given public key \( 5^n \mod p \),
quickly find secret key \( n \)?

(Warning: This is one way to attack the protocol.
Maybe there are better ways.)

Relations to ECC:

1. Some DL techniques also apply to elliptic-curve DL problems.
Use in evaluating security of an elliptic curve.

2. Some techniques don’t apply.
Use in evaluating advantages of elliptic curves compared to multiplication.

3. Tricky: Some techniques have extra applications to some curves.
See Tanja Lange’s talk on Weil descent etc.
The discrete-logarithm problem

Define \( p = 1000003 \).

Easy to prove: \( p \) is prime.

Can we find an integer \( n \in \{1, 2, 3, \ldots, p-1\} \) such that \( 5^n \mod p = 262682 \)?

Easy to prove: \( n \) permutes \( \{1, 2, 3, \ldots, p-1\} \).

So there exists an \( n \) such that \( 5^n \mod p = 262682 \).

Could find \( n \) by brute force.

Is there a faster way?

Typical cryptanalytic application:

Imagine standard \( p = 1000003 \) in the Diffie-Hellman protocol.

User chooses secret key \( n \), publishes \( 5^n \mod p = 262682 \).

Can attacker quickly solve the discrete-logarithm problem?

Given public key \( 5^n \mod p \), quickly find secret key \( n \)?

(Warning: This is one way to attack the protocol.
Maybe there are better ways.)

Relations to ECC:

1. Some DL techniques also apply to elliptic-curve DL problems.
   Use in evaluating security of an elliptic curve.

2. Some techniques don’t apply.
   Use in evaluating advantages of elliptic curves compared to multiplication.

3. Tricky: Some techniques have extra applications to some curves.
   See Tanja Lange’s talk on Weil descent etc.
Typical cryptanalytic application:
Imagine standard $p = 1000003$ in the Diffie-Hellman protocol.

User chooses secret key $n$, publishes $5^n \mod p = 262682$.

Can attacker quickly solve the discrete-logarithm problem?
Given public key $5^n \mod p$, quickly find secret key $n$?

(Warning: This is one way to attack the protocol.
Maybe there are better ways.)

Relations to ECC:
1. Some DL techniques also apply to elliptic-curve DL problems.
Use in evaluating security of an elliptic curve.

2. Some techniques don’t apply.
Use in evaluating advantages of elliptic curves compared to multiplication.

3. Tricky: Some techniques have extra applications to some curves.
See Tanja Lange’s talk on Weil descent etc.
Typical cryptanalytic application:
Imagine standard $p = 1000003$ in the Diffie-Hellman protocol. User chooses secret key $n$, publishes $5^n \mod p = 262682$. Can attacker quickly solve the discrete-logarithm problem? Given public key $5^n \mod p$, quickly find secret key $n$? (Warning: This is one way to attack the protocol. Maybe there are better ways.)

Relations to ECC:
1. Some DL techniques also apply to elliptic-curve DL problems. Use in evaluating security of an elliptic curve.
2. Some techniques don’t apply. Use in evaluating advantages of elliptic curves compared to multiplication.
3. Tricky: Some techniques have extra applications to some curves. See Tanja Lange’s talk on Weil descent etc.

Understanding brute force
Can compute successively
$5^1 \mod p = 5,$
$5^2 \mod p = 25,$
$5^3 \mod p = 125, \ldots$,
$5^{1000002} \mod p = 1.$
At some point we’ll find $n$ with $5^n \mod p = 262682.$
Maximum cost of computation:
$p$ 1 mults by 5 mod $p$;
$p$ 1 nanoseconds on a CPU that does 1 mult/nanosecond.
Relations to ECC:

1. Some DL techniques also apply to elliptic-curve DL problems.
   Use in evaluating security of an elliptic curve.

2. Some techniques don’t apply.
   Use in evaluating advantages of elliptic curves compared to multiplication.

3. Tricky: Some techniques have extra applications to some curves.
   See Tanja Lange’s talk on Weil descent etc.

Understanding brute force

Can compute successively

\[ 5^1 \mod p = 5, \]
\[ 5^2 \mod p = 25, \]
\[ 5^3 \mod p = 125, \]
\[ 5^8 \mod p = 390625, \]
\[ 5^9 \mod p = 953125, \]
\[ 5^{1000002} \mod p = 1. \]

At some point we’ll find key \( n \) with \( 5^n \mod p = 262682. \)

Maximum cost of computation:
\[ \leq p - 1 \text{ mults by 5 mod } p; \]
\[ \leq p - 1 \text{ nanoseconds on a CPU that does 1 mult/nanosecond.} \]
Relations to ECC:

1. Some DL techniques also apply to elliptic-curve DL problems.
   Use in evaluating security of an elliptic curve.

2. Some techniques don't apply.
   Use in evaluating advantages of elliptic curves compared to multiplication.

3. Tricky: Some techniques have extra applications to some curves.
   See Tanja Lange's talk on Weil descent etc.

Understanding brute force

Can compute successively

\[ 5^1 \mod p = 5, \]
\[ 5^2 \mod p = 25, \]
\[ 5^3 \mod p = 125, \ldots, \]
\[ 5^8 \mod p = 390625, \]
\[ 5^9 \mod p = 953122, \ldots, \]
\[ 5^{1000002} \mod p = 1. \]

At some point we’ll find \( n \) with \( 5^n \mod p = 262682. \)

Maximum cost of computation:

\[ \leq p - 1 \text{ mults by } 5 \mod p; \]
\[ \leq p - 1 \text{ nanoseconds on a CPU that does } 1 \text{ mult/nanosecond.} \]
Relations to ECC:

1. Some DL techniques also apply to elliptic-curve DL problems. Use in evaluating security of an elliptic curve.

2. Some techniques don’t apply. Use in evaluating advantages of elliptic curves compared to multiplication.

3. Tricky: Some techniques have extra applications to some curves. See Tanja Lange’s talk on Weil descent etc.

Understanding brute force

Can compute successively

\[5^1 \mod p = 5,\]
\[5^2 \mod p = 25,\]
\[5^3 \mod p = 125, \ldots,\]
\[5^8 \mod p = 390625,\]
\[5^9 \mod p = 953122, \ldots,\]
\[5^{1000002} \mod p = 1.\]

At some point we’ll find \(n\) with \(5^n \mod p = 262682.\)

Maximum cost of computation:

\[\leq p - 1\] mults by 5 mod \(p;\)
\[\leq p - 1\] nanoseconds on a CPU that does 1 mult/nanosecond.
Understanding brute force

Can compute successively
\[ 5^1 \mod p = 5, \]
\[ 5^2 \mod p = 25, \]
\[ 5^3 \mod p = 125, \ldots, \]
\[ 5^8 \mod p = 390625, \]
\[ 5^9 \mod p = 953122, \ldots, \]
\[ 5^{1000002} \mod p = 1. \]

At some point we’ll find \( n \) with \( 5^n \mod p = 262682. \)

Maximum cost of computation:
\[ \leq p - 1 \text{ mults by 5 mod } p; \]
\[ \leq p - 1 \text{ nanoseconds on a CPU that does 1 mult/nanosecond.} \]

This is negligible work for \( p \approx 2^{50} \).

But users can standardize a larger \( p \), making the attack slower.

Attack cost scales linearly:
\[ \approx 2^{50} \text{ mults for } p \approx 2^{50}, \]
\[ \approx 2^{100} \text{ mults for } p \approx 2^{100}, \] etc.

(Not exactly linearly: cost of mults grows with \( p \). But this is a minor effect.)
Relations to ECC:

1. Some DL techniques also apply to elliptic-curve DL problems. Use in evaluating security of an elliptic curve.

2. Some techniques don’t apply. Use in evaluating advantages of elliptic curves compared to multiplication.

3. Tricky: Some techniques have extra applications to some curves. See Tanja Lange’s talk on Weil descent etc.

Understanding brute force

Can compute successively

\[ 5^1 \mod p = 5, \]
\[ 5^2 \mod p = 25, \]
\[ 5^3 \mod p = 125, \ldots, \]
\[ 5^8 \mod p = 390625, \]
\[ 5^9 \mod p = 953122, \ldots, \]
\[ 5^{1000002} \mod p = 1. \]

At some point we’ll find \( n \) with \( 5^n \mod p = 262682. \)

Maximum cost of computation:

\[ \leq p - 1 \text{ mults by } 5 \mod p; \]
\[ \leq p - 1 \text{ nanoseconds on a CPU that does } 1 \text{ mult/nanosecond.} \]

This is negligible work for \( p \approx 2^{20}. \)

But users can standardize a larger \( p \), making the attack slower.

Attack cost scales linearly:

\( \approx 2^{50} \text{ mults for } p \approx 2^{50}, \)
\( \approx 2^{100} \text{ mults for } p \approx 2^{100}, \) etc.

(Not exactly linearly: cost of mults grows with \( p \). But this is a minor effect.)
### Understanding brute force

Can compute successively:

\[
\begin{align*}
5^1 & \mod p = 5, \\
5^2 & \mod p = 25, \\
5^3 & \mod p = 125, \ldots, \\
5^8 & \mod p = 390625, \\
5^9 & \mod p = 953122, \ldots, \\
5^{1000002} & \mod p = 1.
\end{align*}
\]

At some point we’ll find \( n \) with \( 5^n \mod p = 262682. \)

Maximum cost of computation:

\[
\leq p - 1 \text{ mults by } 5 \mod p;
\]

\[
\leq p - 1 \text{ nanoseconds on a CPU that does } 1 \text{ mult/nanosecond.}
\]

This is negligible work for \( p \approx 2^{20} \).

But users can standardize a larger \( p \), making the attack slower.

Attack cost scales linearly:

\[
\approx 2^{50} \text{ mults for } p \approx 2^{50},
\]

\[
\approx 2^{100} \text{ mults for } p \approx 2^{100}, \text{ etc.}
\]

(Not exactly linearly: cost of mults grows with \( p \).)

But this is a minor effect.)
Understanding brute force

Can compute successively

\[ 5^1 \mod p = 5, \]
\[ 5^2 \mod p = 25, \]
\[ 5^3 \mod p = 125, \ldots, \]
\[ 5^8 \mod p = 390625, \]
\[ 5^9 \mod p = 953122, \ldots, \]
\[ 5^{1000002} \mod p = 1. \]

At some point we’ll find \( n \) with \( 5^n \mod p = 262682. \)

Maximum cost of computation:
\[ \leq p - 1 \text{ mults by } 5 \mod p; \]
\[ \leq p - 1 \text{ nanoseconds on a CPU that does 1 mult/nanosecond.} \]

This is negligible work for \( p \approx 2^{20}. \)

But users can standardize a larger \( p, \) making the attack slower.

Attack cost scales linearly:
\[ \approx 2^{50} \text{ mults for } p \approx 2^{50}, \]
\[ \approx 2^{100} \text{ mults for } p \approx 2^{100}, \text{ etc.} \]

(Not exactly linearly: cost of mults grows with \( p. \)

But this is a minor effect.)
Understanding brute force

Can compute successively

\[ 5^1 \mod p = 5, \]
\[ 5^2 \mod p = 25, \]
\[ 5^3 \mod p = 125, \ldots, \]
\[ 5^8 \mod p = 390625, \]
\[ 5^9 \mod p = 953125, \ldots, \]
\[ 5^{1000002} \mod p = 1. \]

At some point we’ll find \( n \) with \( 5^n \mod p = 262682. \)

Maximum cost of computation:
\[ p \times 1 \text{ mults by } 5 \mod p; \]
\[ p \times 1 \text{ nanoseconds on a CPU that does } 1 \text{ mult/nanosecond.} \]

This is negligible work for \( p \approx 2^{20}. \)

But users can standardize a larger \( p, \) making the attack slower.

Attack cost scales linearly:
\[ \approx 2^{50} \text{ mults for } p \approx 2^{50}, \]
\[ \approx 2^{100} \text{ mults for } p \approx 2^{100}, \text{ etc.} \]

(Not exactly linearly: cost of mults grows with \( p. \) But this is a minor effect.)

But users can standardize a larger \( p, \) making the attack slower.

Attack cost scales linearly:
\[ \approx 2^{50} \text{ mults for } p \approx 2^{50}, \]
\[ \approx 2^{100} \text{ mults for } p \approx 2^{100}, \text{ etc.} \]

(Not exactly linearly: cost of mults grows with \( p. \) But this is a minor effect.)

This is negligible work for \( p \approx 2^{20}. \)

But users can standardize a larger \( p, \) making the attack slower.

Attack cost scales linearly:
\[ \approx 2^{50} \text{ mults for } p \approx 2^{50}, \]
\[ \approx 2^{100} \text{ mults for } p \approx 2^{100}, \text{ etc.} \]

(Not exactly linearly: cost of mults grows with \( p. \) But this is a minor effect.)

That’s pointless. We can apply “random self-reduction”:
choose random \( r, \) say 726379;
compute \( 5^r \mod p = 515040; \)
compute \( 5^r \cdot 5^n \mod p \) as \((515040 \cdot (5^n \mod p)) \mod p; \)
compute discrete log;
subtract \( r \mod p \) \(- 1; \) obtain \( n. \)
This is negligible work for \( p \approx 2^{20} \).

But users can standardize a larger \( p \), making the attack slower.

Attack cost scales linearly:
\( \approx 2^{50} \) mults for \( p \approx 2^{50} \),
\( \approx 2^{100} \) mults for \( p \approx 2^{100} \), etc.

(Not exactly linearly: cost of mults grows with \( p \). But this is a minor effect.)

Computation has a good chance of finishing earlier.

Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost;

“So users should choose large \( n \).”

That’s pointless. We can apply “random self-reduction”:
choose random \( r \), say 262682.
compute \( 5^r \mod p \);
compute \( 5^r 5^n \mod p \) as \((515040 \cdot (5^n \mod p)) \mod p\);
compute discrete log;
subtract \( r \mod p - 1\); obtain \( n \).
This is negligible work for \( p \approx 2^{20} \).

But users can standardize a larger \( p \), making the attack slower.

Attack cost scales linearly:
\[ \approx 2^{50} \text{ mults for } p \approx 2^{50} , \]
\[ \approx 2^{100} \text{ mults for } p \approx 2^{100} , \text{ etc.} \]

(Not exactly linearly: cost of mults grows with \( p \). But this is a minor effect.)

Computation has a good chance of finishing earlier.

Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost; etc.

“So users should choose large \( p \).”

That’s pointless. We can apply “random self-reduction”:
choose random \( r \), say 726379;
compute \( 5^r \mod p = 515040 \);
compute \( 5^r \cdot 5^n \mod p \) as \((515040 \cdot (5^n \mod p)) \mod p \);
compute discrete log;
subtract \( r \mod p - 1 \); obtain \( n \).
This is negligible work for $p \approx 2^{20}$.

But users can standardize a larger $p$, making the attack slower.

Attack cost scales linearly:

$\approx 2^{50}$ mults for $p \approx 2^{50}$,

$\approx 2^{100}$ mults for $p \approx 2^{100}$, etc.

(Not exactly linearly: cost of mults grows with $p$.
But this is a minor effect.)

Computation has a good chance of finishing earlier.
Chance scales linearly:

$1/2$ chance of $1/2$ cost;

$1/10$ chance of $1/10$ cost; etc.

“So users should choose large $n$.”

That’s pointless. We can apply “random self-reduction”:
choose random $r$, say 726379;
compute $5^r \mod p = 515040$;
compute $5^r 5^n \mod p$ as $(515040 \cdot (5^n \mod p)) \mod p$;
compute discrete log;
subtract $r \mod p - 1$; obtain $n$. 
This is negligible work for $p^{2^{20}}$.

Users can standardize a larger $p$, making the attack slower.

Cost scales linearly: $2^{50}$ mults for $p \approx 2^{50}$,
$2^{100}$ mults for $p \approx 2^{100}$, etc.

(Not exactly linearly: cost of mults grows with $p$.
But this is a minor effect.)

Computation has a good chance of finishing earlier.
Chance scales linearly:
$1/2$ chance of $1/2$ cost;
$1/10$ chance of $1/10$ cost; etc.

“So users should choose large $n$.”

That’s pointless. We can apply “random self-reduction”:
choose random $r$, say 726379;
compute $5^r \mod p = 515040$;
compute $5^r 5^n \mod p$ as $(515040 \cdot (5^n \mod p)) \mod p$;
compute discrete log;
subtract $r \mod p - 1$; obtain $n$.

Computation can be parallelized.
One low-cost chip can run many parallel searches.
Example, $2^{6}$: one chip, $2^{10}$ cores on the chip, each $2^{30}$ mults/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, $2^{30}$: $2^{24}$ chips, so $2^{34}$ cores, so $2^{64}$ mults/second, so $2^{89}$ mults/year.
This is negligible work for $p^{20}$. But users can standardize a larger $p$, making the attack slower.

Attack cost scales linearly:
- $2^{50}$ mults for $p^{250}$,
- $2^{100}$ mults for $p^{2100}$, etc.

(Not exactly linearly: cost of mults grows with $p$. But this is a minor effect.)

Computation has a good chance of finishing earlier.
Chance scales linearly:
- $1/2$ chance of $1/2$ cost;
- $1/10$ chance of $1/10$ cost; etc.

“So users should choose large $n$.”

That’s pointless. We can apply “random self-reduction”:
- choose random $r$, say 726379;
- compute $5^r \mod p = 515040$;
- compute $5^r 5^n \mod p$ as $(515040 \cdot (5^n \mod p)) \mod p$;
- compute discrete log;
- subtract $r \mod p - 1$; obtain $n$.

Computation can be parallelized.
One low-cost chip can run many parallel searches.
Example, $2^6 \in$: one chip, $2^{10}$ cores on the chip, each $2^{30}$ mults/second?
Maybe; see SHARC for detailed cost analyses.

Attacker can run many parallel chips.
Example, $2^{30} \in$: 2$^{24}$ chips, so $2^{34}$ cores, so $2^{64}$ mults/second, so $2^{89}$ mults/year.
Computation has a good chance of finishing earlier.
Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost; etc.

“So users should choose large $n$.”

That’s pointless. We can apply “random self-reduction”:
choose random $r$, say 726379;
compute $5^r \mod p = 515040$;
compute $5^r 5^n \mod p$ as
$(515040 \cdot (5^n \mod p)) \mod p$;
compute discrete log;
subtract $r \mod p - 1$; obtain $n$.

Computation can be parallelized.
One low-cost chip can run many parallel searches.
Example, $2^6 \in$: one chip, $2^{10}$ cores on the chip, each $2^{30}$ mults/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, $2^{30} \in$: $2^{24}$ chips, so $2^{34}$ cores,
so $2^{64}$ mults/second,
so $2^{89}$ mults/year.
Computation has a good chance of finishing earlier.
Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost; etc.

“So users should choose large $n$.”

That’s pointless. We can apply “random self-reduction”:
choose random $r$, say 726379;
compute $5^r \mod p = 515040$;
compute $5^r 5^n \mod p$ as $(515040 \cdot (5^n \mod p)) \mod p$;
compute discrete log;
subtract $r \mod p - 1$; obtain $n$.

Computation can be parallelized.
One low-cost chip can run many parallel searches.
Example, $2^6 \in$: one chip, $2^{10}$ cores on the chip, each $2^{30}$ mults/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, $2^{30} \in$: $2^{24}$ chips, so $2^{34}$ cores, so $2^{64}$ mults/second, so $2^{89}$ mults/year.
Computation has a good chance of finishing earlier.

Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost; etc.

Users should choose large $n$.

That's pointless. We can apply "random self-reduction":

Choose random $r$, say 726379;

Compute $5^r \mod p = 515040$;

Compute $5^r 5^n \mod p$ as
$(5^n \mod p)(5^r \mod p)$ mod $p$;

Compute discrete log;

Subtract $r \mod p - 1$; obtain $n$.

Computation can be parallelized.

One low-cost chip can run many parallel searches.

Example, $2^6 \in$: one chip, $2^{10}$ cores on the chip, each $2^{30}$ mults/second?

Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.

Example, $2^{30} \in$: $2^{24}$ chips, $2^{34}$ cores, $2^{64}$ mults/second, $2^{89}$ mults/year.

Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets $5^n_1 \mod p, 5^n_2 \mod p, \ldots$:

Can find all of $n_1, n_2, \ldots, n_{100}$ with $\leq p$ mults.

Simplest approach: First build a sorted table containing $5^n_1 \mod p, 5^n_2 \mod p, \ldots$.

Then check table for $5^1 \mod p, 5^2 \mod p, \ldots$.
Computation can be parallelized.

One low-cost chip can run many parallel searches.
Example, $2^6 \in$: one chip, $2^{10}$ cores on the chip, each $2^{30}$ mults/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, $2^{30} \in$: $2^{24}$ chips, so $2^{34}$ cores, so $2^{64}$ mults/second, so $2^{89}$ mults/year.

Multiple targets and giant steps
Computation can be applied to many targets at once.
Given 100 DL targets $5^{n_1} \mod p$, \ldots, $5^{n_{100}} \mod p$:
Can find all of $n_1$, \ldots, $n_{100}$ with $\leq p - 1$ mults mod $p$.

Simplest approach: First build a sorted table containing $5^{n_1} \mod p$, \ldots, $5^{n_{100}} \mod p$.
Then check table for $5^1 \mod p$, $5^2 \mod p$, etc.
Computation can be parallelized.

One low-cost chip can run many parallel searches.
Example, $2^6 \in$: one chip, $2^{10}$ cores on the chip, each $2^{30}$ mults/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, $2^{30} \in$: $2^{24}$ chips, so $2^{34}$ cores, so $2^{64}$ mults/second, so $2^{89}$ mults/year.

Multiple targets and giant steps
Computation can be applied to many targets at once.

Given 100 DL targets $5^{n_1} \mod p$, $5^{n_2} \mod p$, \ldots, $5^{n_{100}} \mod p$.
Can find all of $n_1$, $n_2$, \ldots, $n_{100}$ with $\leq p - 1$ mults $\mod p$.

Simplest approach: First build a sorted table containing $5^{n_1} \mod p$, \ldots, $5^{n_{100}} \mod p$.
Then check table for $5^1 \mod p$, $5^2 \mod p$, etc.
Computation can be parallelized.
One low-cost chip can run many parallel searches.
Example, $2^6$: one chip, $2^{10}$ cores on the chip, each $2^{30}$ mults/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, $2^{30} = 2^{24}$ chips, so $2^{34}$ cores, so $2^{64}$ mults/second, so $2^{89}$ mults/year.

Multiple targets and giant steps
Computation can be applied to many targets at once.
Given 100 DL targets $5^{n_1} \mod p$, $5^{n_2} \mod p$, \ldots, $5^{n_{100}} \mod p$:
Can find all of $n_1, n_2, \ldots, n_{100}$ with $\leq p - 1$ mults $\mod p$.

Simplest approach: First build a sorted table containing $5^{n_1} \mod p$, \ldots, $5^{n_{100}} \mod p$.
Then check table for $5^1 \mod p$, $5^2 \mod p$, etc.
Computation can be parallelized. One low-cost chip can run many parallel searches.

Example, $2^6 \in$: one chip, $2^{30} \in$: 24 chips, $2^{64} \in$: 64 cores, 2^{89} \in: 64 \times 2^{89} \text{mults/year}?

see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.

Example, $2^{30} \in$: 24 chips, $2^{64} \in$: 64 cores, $2^{89} \in$: 64 \times 2^{89} \text{mults/year}.

Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets $5^{n_1} \mod p$, $5^{n_2} \mod p$, $\ldots$, $5^{n_{100}} \mod p$:

Can find all of $n_1$, $n_2$, $\ldots$, $n_{100}$ with $\leq p - 1 \text{mults mod } p$.

Simplest approach: First build a sorted table containing $5^{n_1} \mod p$, $\ldots$, $5^{n_{100}} \mod p$.

Then check table for $5^1 \mod p$, $5^2 \mod p$, etc.

Interesting consequence #1: Solving all 100 DL problems isn't much harder than solving one DL problem.

Interesting consequence #2: Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first $n_i$?

Typically $(p - 1) = 100 \text{mults}$. 
Computation can be parallelized. One low-cost chip can run many parallel searches. Example, $2^{6} = 64$ cores on the chip, each $2^{30}$ mults/second? Maybe; see SHARCS workshops for detailed cost analyses. Attacker can run many parallel chips. Example, $2^{30} = 2^{24}$ chips, so $2^{34}$ cores, so $2^{64}$ mults/second, so $2^{89}$ mults/year.

Multiple targets and giant steps

Computation can be applied to many targets at once. Given 100 DL targets $5^{n_{1}} \mod p$, $5^{n_{2}} \mod p$, ..., $5^{n_{100}} \mod p$: Can find all of $n_{1}, n_{2}, \ldots, n_{100}$ with $\leq p - 1$ mults mod $p$.

Simplest approach: First build a sorted table containing $5^{n_{1}} \mod p$, ..., $5^{n_{100}} \mod p$. Then check table for $5^{1} \mod p$, $5^{2} \mod p$, etc.

Interesting consequence #1: Solving all 100 DL problems isn't much harder than solving one DL problem.

Interesting consequence #2: Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first $n_{i}$?

Typically $(p - 1) = 100$ mults.
Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets $5^{n_1} \mod p$, $5^{n_2} \mod p$, \ldots, $5^{n_{100}} \mod p$:
Can find all of $n_1, n_2, \ldots, n_{100}$ with $\leq p - 1$ mults mod $p$.

Simplest approach: First build a sorted table containing $5^{n_1} \mod p$, \ldots, $5^{n_{100}} \mod p$.
Then check table for $5^1 \mod p$, $5^2 \mod p$, etc.

Interesting consequence #1:
Solving all 100 DL problems isn’t much harder than solving one DL problem.

Interesting consequence #2:
Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first $n_i$?
Typically $\approx (p - 1)/100$ mults.
Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets $5^{n_1} \mod p$, $5^{n_2} \mod p$, \ldots, $5^{n_{100}} \mod p$:
Can find all of $n_1, n_2, \ldots, n_{100}$ with $\leq p - 1$ mults mod $p$.

Simplest approach: First build a sorted table containing $5^{n_1} \mod p$, \ldots, $5^{n_{100}} \mod p$.
Then check table for $5^1 \mod p$, $5^2 \mod p$, etc.

Interesting consequence #1:
Solving all 100 DL problems isn’t much harder than solving one DL problem.

Interesting consequence #2:
Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first $n_i$?
Typically $\approx (p - 1)/100$ mults.
targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets 5^{n_1} \mod p, 5^{n_2} \mod p, \ldots, 5^{n_{100}} \mod p:

\begin{align*}
\text{find all of } n_1, n_2, \ldots, n_{100} \\
\text{with } p - 1 \text{ mults } \mod p.
\end{align*}

Simplest approach: First build a sorted table containing 5^{n_1} \mod p, \ldots, 5^{n_{100}} \mod p.
Then check table for 5^{n_1} \mod p, 5^{n_2} \mod p, etc.

Interesting consequence #1:
Solving all 100 DL problems isn't much harder than solving one DL problem.

Interesting consequence #2:
Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first n_i?
Typically \( (p - 1)/100 \) mults.

Can use random self-reduction to turn a single target into multiple targets.

Given 5^{n_1} \mod p:
Choose random \( r_1, \ldots, r_{100} \).

Compute 5^{r_1^{n_1} \mod p}, 5^{r_2^{n_1} \mod p}, \ldots, 5^{r_{100}^{n_1} \mod p}.

Solve these 100 DL problems.
Typically \( p - 1 \) mults to find at least one \( r_i + n_1 \mod p \),
immediately revealing \( r_i + n_1 \mod p \).
Multiple targets and giant steps

Computation can be applied to many targets at once. Given 100 DL targets:

\[
\begin{align*}
5^n & \pmod{p} \\
5^{n_1} & \pmod{p} \\
5^{n_2} & \pmod{p} \\
& \vdots \\
5^{n_{100}} & \pmod{p} \\
\end{align*}
\]

Can find all of \(n_1; n_2; \ldots; n_{100}\) with \(p - 1\) mults \(\pmod{p}\).

Simplest approach: First build a sorted table containing

\[
\begin{align*}
5^n & \pmod{p} \\
5^{n_1} & \pmod{p} \\
5^{n_2} & \pmod{p} \\
& \vdots \\
5^{n_{100}} & \pmod{p} \\
\end{align*}
\]

Then check table for

\[
\begin{align*}
5^n & \pmod{p} \\
5^{n_1} & \pmod{p} \\
5^{n_2} & \pmod{p} \\
& \vdots \\
5^{n_{100}} & \pmod{p} \\
\end{align*}
\]

Interesting consequence #1: Solving all 100 DL problems isn’t much harder than solving one DL problem.

Interesting consequence #2: Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first \(n_i\)? Typically \(\approx (p - 1)/100\) mults.

Can use random self-reduction to turn a single target into multiple targets.

Given \(5^n \pmod{p}\):

Choose random \(r_1, r_2, \ldots, r_{100}\).

Compute \(5^{r_1n} \pmod{p}\), \(5^{r_2n} \pmod{p}\), etc.

Solve these 100 DL problems. Typically \(\approx (p - 1)/100\) to find at least one \(r_i + n \pmod{p} - 1\), immediately revealing \(n\).
Multiple targets and giant steps

Computation can be applied to many targets at once. Given 100 DL targets
\[ 5^n \mod p, \quad n = 1, 2, \ldots, 100 \]

Can find all of \( n_1, n_2, \ldots, n_{100} \) with \( p-1 \) mults mod \( p \).

Simplest approach: First build a sorted table containing
\[ 5^n \mod p, \quad n = 1, 2, \ldots, 100 \]

Then check table for \( 5^1 \mod p, 5^2 \mod p \), etc.

Interesting consequence #1:
Solving all 100 DL problems isn’t much harder than solving one DL problem.

Interesting consequence #2:
Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first \( n_i \)? Typically \( \approx (p - 1)/100 \) mults.

Can use random self-reduction to turn a single target
into multiple targets.

Given \( 5^n \mod p \):
Choose random \( r_1, r_2, \ldots, r_{100} \): \( 1, 2, \ldots, p-1 \).
Compute \( 5^{r_1} n \mod p \), \( 5^{r_2} n \mod p \), etc.

Solve these 100 DL problems. Typically \( \approx (p - 1)/100 \) mults to find at least one \( r_i + n \mod p - 1 \), immediately revealing \( n \).
Interesting consequence #1: Solving all 100 DL problems isn’t much harder than solving one DL problem.

Interesting consequence #2: Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first \( n_i \)? Typically \( \approx (p - 1)/100 \) mults.

Can use random self-reduction to turn a single target into multiple targets.

Given \( 5^n \mod p \):
Choose random \( r_1, r_2, \ldots, r_{100} \).
Compute \( 5^{r_1}5^n \mod p \), \( 5^{r_2}5^n \mod p \), etc.
Solve these 100 DL problems. Typically \( \approx (p - 1)/100 \) mults to find at least one \( r_i + n \mod p - 1 \), immediately revealing \( n \).
Interesting consequence #1: Solving all 100 DL problems isn’t much harder than solving one DL problem.

Interesting consequence #2: Solving at least one out of 100 DL problems is much easier than solving one DL problem.

Typically \((p - 1) = 100\) mults.

Can use random self-reduction to turn a single target into multiple targets.

Given \(5^n \mod p\):
Choose random \(r_1, r_2, \ldots, r_{100}\).
Compute \(5^{r_1}5^n \mod p\), \(5^{r_2}5^n \mod p\), etc.
Solve these 100 DL problems.
Typically \(\approx (p - 1)/100\) mults to find at least one \(r_i + n \mod p - 1\), immediately revealing \(n\).

Also spent some mults to compute each \(5^{r_i} \mod p\): \(\approx \lg p\) mults for each \(i\).

Faster: Choose \(r_i = ir_1\) with \(r_1 = \ldots = r_{100}\).
Compute \(5^{r_1}5^n \mod p\), \(5^{2r_1}5^n \mod p\), \(5^{3r_1}5^n \mod p\), etc.
Just 1 mult for each new \(i\).
\(\approx 100 + \lg p + (p - 1) = 100\) mults to find \(n\) given \(5^n \mod p\).
Interesting consequence #1: Solving all 100 DL problems isn't much harder than solving one DL problem.

Interesting consequence #2: Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first \( n \)?

Typically \( (p - 1)/100 \) mults.

Can use random self-reduction to turn a single target into multiple targets.

Given \( 5^n \mod p \):
Choose random \( r_1, r_2, \ldots, r_{100} \).
Compute \( 5^{r_1}n \mod p \), \( 5^{r_2}n \mod p \), etc.

Solve these 100 DL problems.
Typically \( \approx (p - 1)/100 \) mults to find at least one \( r_i + n \mod p - 1 \), immediately revealing \( n \).

Also spent some mults to compute each \( 5^{r_i}n \mod p \): \( \approx \lg p \) mults for each \( i \).

Faster: Choose \( r_i = ir_1 \) with \( r_1 \approx (p - 1)/100 \).
Compute \( 5^{r_1}n \mod p \); \( 5^{r_1}n \mod p \); \( 5^{2r_1}n \mod p \); \( 5^{3r_1}n \mod p \); etc.
Just 1 mult for each new \( i \).
\( \approx 100 + \lg p + (p - 1)/100 \) mults to find \( n \) given \( 5^n \mod p \).
Interesting consequence #1:
Solving all 100 DL problems
isn't much harder than
solving one DL problem.

Interesting consequence #2:
Solving at least one out of 100 DL problems
is much easier than solving one DL problem.

When did this computation
find its first $n_i$?

Typically $(p-1) = 100$ mults.

Can use random self-reduction
to turn a single target
into multiple targets.

Given $5^n \mod p$:
Choose random $r_1, r_2, \ldots, r_{100}$.
Compute $5^{r_1}5^n \mod p$, $5^{r_2}5^n \mod p$, etc.
Solve these 100 DL problems.
Typically $\approx (p-1)/100$ mults
to find at least one
$r_i + n \mod p - 1$,
immediately revealing $n$.

Also spent some mults
to compute each $5^{r_i} \mod p$:
$\approx \lg p$ mults for each $i$.

Faster: Choose $r_i = ir_1$
with $r_1 \approx (p - 1)/100$.
Compute $5^{r_1} \mod p$;
$5^{r_1}5^n \mod p$;
$5^{2r_1}5^n \mod p$;
$5^{3r_1}5^n \mod p$; etc.
Just 1 mult for each new $i$.
$\approx 100 + \lg p + (p - 1)/100$
to find $n$ given $5^n \mod p$. 
Can use random self-reduction to turn a single target into multiple targets.

Given $5^n \mod p$:
Choose random $r_1, r_2, \ldots, r_{100}$.
Compute $5^{r_1}5^n \mod p$,
$5^{r_2}5^n \mod p$, etc.
Solve these 100 DL problems.
Typically $\approx (p - 1)/100$ mults to find at least one
$r_i + n \mod p - 1$,
immediately revealing $n$.

Also spent some mults to compute each $5^{r_i} \mod p$:
$\approx \lg p$ mults for each $i$.
Faster: Choose $r_i = ir_1$ with $r_1 \approx (p - 1)/100$.
Compute $5^{r_1} \mod p$;
$5^{r_1}5^n \mod p$;
$5^{2r_1}5^n \mod p$;
$5^{3r_1}5^n \mod p$; etc.
Just 1 mult for each new $i$.
$\approx 100 + \lg p + (p - 1)/100$ mults to find $n$ given $5^n \mod p$. 
Can use random self-reduction to turn a single target into multiple targets.

Given $5^n \mod p$:

Choose random $r_1, r_2, \ldots, r_{100}$.

Compute $5^{r_1}5^n \mod p$,
$5^{r_2}5^n \mod p$, etc.

Solve these 100 DL problems.

Typically $(p - 1) = 100$ mults to find at least one $r_i + n \mod p = 1$, immediately revealing $n$.

Also spent some mults to compute each $5^{r_i} \mod p$:

- $\approx \lg p$ mults for each $i$.

Faster: Choose $r_i = ir_1$ with $r_1 \approx (p - 1)/100$.

Compute $5^{r_1} \mod p$;
$5^{r_1}5^n \mod p$;
$5^{2r_1}5^n \mod p$;
$5^{3r_1}5^n \mod p$; etc.

Just 1 mult for each new $i$.

$\approx 100 + \lg p + (p - 1)/100$ mults to find $n$ given $5^n \mod p$.

Faster: Increase 100 to $p$.

Only $\approx 2p$ mults to solve one DL problem!

“Shanks baby-step-giant-step discrete-logarithm algorithm.”

Example: $p = 1000003$, $5^n \mod p = 262682$.

Compute $5^{1024} \mod p = 58588$.

Then compute 1000 targets:
$5^{1024}5^n \mod p = 966849$,
$5^{2 \cdot 1024}5^n \mod p = 579277$,
$5^{3 \cdot 1024}5^n \mod p = 579062$, : : : ,
$5^{1000 \cdot 1024}5^n \mod p = 321705$. 
Can use random self-reduction to turn a single target into multiple targets.

Given $5^n \mod p$:
Choose random $r_1, r_2, \ldots, r_{100}$.

Compute $5^{r_1} 5^n \mod p$,
$5^{r_2} 5^n \mod p$, etc.
Solve these 100 DL problems.

Typically $(p - 1)/100 = 100$ mults to find at least one $r_i + n \mod p = 1$,
immediately revealing $n$.

Also spent some mults to compute each $5^{r_i} \mod p$:
$\approx \lg p$ mults for each $i$.

Faster: Choose $r_i = i r_1$
with $r_1 \approx (p - 1)/100$.

Compute $5^{r_1} \mod p$;
$5^{r_1} 5^n \mod p$;
$5^{2r_1} 5^n \mod p$;
$5^{3r_1} 5^n \mod p$; etc.

Just 1 mult for each new $i$.

$\approx 100 + \lg p + (p - 1)/100$ mults to find $n$ given $5^n \mod p$.

Faster: Increase 100 to $p$.

Only $\approx 2\sqrt{p}$ mults to solve one DL problem!

“Shanks baby-step-giant-step discrete-logarithm algorithm.”

Example: $p = 1000003$,
$5^n \mod p = 262682$.

Compute $5^{1024} \mod p$.

Then compute 1000 targets:
$5^{1024} 5^n \mod p = 966849$,
$5^{2 \cdot 1024} 5^n \mod p = 579277$,
$5^{3 \cdot 1024} 5^n \mod p = 579062$, ...
$5^{1000 \cdot 1024} 5^n \mod p = 321705$. 

Also spent some mults to compute each $5^{r_i} \mod p$:
$\approx \lg p$ mults for each $i$.

Faster: Increase 100 to $p$.

Only $\approx 2\sqrt{p}$ mults to solve one DL problem!

“Shanks baby-step-giant-step discrete-logarithm algorithm.”

Example: $p = 1000003$,
$5^n \mod p = 262682$.

Compute $5^{1024} \mod p$.

Then compute 1000 targets:
$5^{1024} 5^n \mod p = 966849$,
$5^{2 \cdot 1024} 5^n \mod p = 579277$,
$5^{3 \cdot 1024} 5^n \mod p = 579062$, ...
$5^{1000 \cdot 1024} 5^n \mod p = 321705$. 

Also spent some mults to compute each $5^{r_i} \mod p$:
$\approx \lg p$ mults for each $i$.
Can use random self-reduction to turn a single target into multiple targets.

Given \( n \mod p \):

Choose random \( r_1, r_2, \ldots, r_{100} \).

Compute \( 5^{r_1}n \mod p \), \( 5^{r_2}n \mod p \), etc.

Solve these 100 DL problems.

Typically \((p - 1) = 100\) mults to find at least one \( r_i + n \mod p \), immediately revealing \( n \).

Also spent some mults to compute each \( 5^{r_i} \mod p \):

\( \approx \lg p \) mults for each \( i \).

Faster: Choose \( r_i = i r_1 \) with \( r_1 \approx (p - 1)/100 \).

Compute \( 5^{r_1} \mod p \); \( 5^{r_1}n \mod p \); \( 5^{2r_1}n \mod p \); \( 5^{3r_1}n \mod p \); etc.

Just 1 mult for each new \( i \).

\( \approx 100 + \lg p + (p - 1)/100 \) mults to find \( n \) given \( 5^n \mod p \).

Faster: Increase 100 to \( \approx \sqrt{p} \).

Only \( \approx 2\sqrt{p} \) mults to solve one DL problem!

“Shanks baby-step-giant-step discrete-logarithm algorithm.”

Example: \( p = 1000003 \),\( 5^n \mod p = 262682 \).

Compute \( 5^{1024} \mod p = 58588 \).

Then compute 1000 targets:

\( 5^{1024}5^n \mod p = 966849 \), \( 5^{2\cdot1024}5^n \mod p = 579277 \), \( 5^{3\cdot1024}5^n \mod p = 579062 \), \( 5^{1000\cdot1024}5^n \mod p = 321707 \).
Also spent some mults
to compute each \(5^ri \mod p\):
\[\approx \lg p\] mults for each \(i\).

Faster: Choose \(r_i = ir_1\)
with \(r_1 \approx (p - 1)/100\).
Compute \(5^{r_1} \mod p\);
\(5^{1r_1 5^n} \mod p\);
\(5^{2r_1 5^n} \mod p\);
\(5^{3r_1 5^n} \mod p\); etc.
Just 1 mult for each new \(i\).
\[\approx 100 + \lg p + (p - 1)/100\] mults
to find \(n\) given \(5^n \mod p\).

Faster: Increase 100 to \(\approx \sqrt{p}\).
Only \(\approx 2\sqrt{p}\) mults
to solve one DL problem!

“Shanks baby-step-giant-step
discrete-logarithm algorithm.”

Example: \(p = 1000003\),
\(5^n \mod p = 262682\).
Compute \(5^{1024} \mod p = 58588\).
Then compute 1000 targets:
\(5^{1024 5^n} \mod p = 966849,\)
\(5^{2\cdot 1024 5^n} \mod p = 579277,\)
\(5^{3\cdot 1024 5^n} \mod p = 579062, \ldots,\)
\(5^{1000\cdot 1024 5^n} \mod p = 321705.\)
Also spent some mults to compute each $5^r_i \mod p$:
$\log p$ mults for each $i$.
Faster: Choose $r_i = i r_1 \approx (p - 1)/100$.
Compute $5^{r_1} \mod p$;
$5^{r_1} \mod p$;
... $5^{r_1} \mod p$; etc.
Just 1 mult for each new $i$.

$100 + \log p + (p - 1)/100$ mults to solve one DL problem!

“Shanks baby-step-giant-step discrete-logarithm algorithm.”

Example: $p = 1000003$,
$5^n \mod p = 262682$.
Compute $5^{1024} \mod p = 58588$.
Then compute 1000 targets:
$5^{1024} 5^n \mod p = 966849$,
$5^{2 \cdot 1024} 5^n \mod p = 579277$,
$5^{3 \cdot 1024} 5^n \mod p = 579062$, \ldots,
$5^{1000 \cdot 1024} 5^n \mod p = 321705$.

Build a sorted table of targets:

- $2573 = 5^{430} 1024 \mod p$,
- $3371 = 5^{192} 1024 \mod p$,
- $3593 = 5^{626} 1024 \mod p$,
- $4960 = 5^{663} 1024 \mod p$,
- $5218 = 5^{376} 1024 \mod p$,
- ... $999675 = 5^{344} 1024 \mod p$.

Look up $5^{755} \mod p$; $5^{332} 1024 \mod p$ in the table of targets; so $755 = 332 1024 + n \mod p$.
Deduce $n = 660789$. 

Faster: Increase 100 to $\approx \sqrt{p}$.
Only $\approx 2 \sqrt{p}$ mults to compute each $5^r_i \mod p$.
Faster: Increase 100 to $\approx \sqrt{p}$. Only $\approx 2\sqrt{p}$ mults to solve one DL problem!

“Shanks baby-step-giant-step discrete-logarithm algorithm.”

Example: $p = 1000003$, $5^n \mod p = 262682$.

Compute $5^{1024} \mod p = 58588$.

Then compute 1000 targets:
$5^{1024}5^n \mod p = 966849$, $5^{2\cdot1024}5^n \mod p = 579277$, $5^{3\cdot1024}5^n \mod p = 579062$, $5^{4\cdot1024}5^n \mod p = 478599$, $5^{5\cdot1024}5^n \mod p = 321705$, $5^{6\cdot1024}5^n \mod p = 66091$, $5^{7\cdot1024}5^n \mod p = 262682$.

Build a sorted table of targets:
$2573 = 5^{430}\cdot10245^n$, $3371 = 5^{192}\cdot10245^n$, $3593 = 5^{626}\cdot10245^n$, $4960 = 5^{663}\cdot10245^n$, $5218 = 5^{376}\cdot10245^n$, $999675 = 5^{344}\cdot10245^n$.

Look up $5^1 \mod p$, $5^3 \mod p$, etc. in the table of targets.

$5^{755} \mod p = 966603$; find $966603 = 5^{332}\cdot10245^n$ in the table of targets, so $755 = 332\cdot1024 + n \mod p$; deduce $n = 660789$. 

Also spent some mults to compute each $5^r_i \mod p$: $\lg p$ mults for each $i$.

Faster: Choose $r_i = ir_1$ with $r_1 \equiv (p_1)^{-1} \mod p$.

Compute $5^{r_1} \mod p$, $5^{2r_1} \mod p$, $5^{3r_1} \mod p$, etc.

Just 1 mult for each new $i$.

$(100 + \lg p + (p_1)) = 100$ mults to find $n$, given $5^n \mod p$. 

Build a sorted table of targets: $2573 = 5^{430}\cdot10245^n$, $3371 = 5^{192}\cdot10245^n$, $3593 = 5^{626}\cdot10245^n$, $4960 = 5^{663}\cdot10245^n$, $5218 = 5^{376}\cdot10245^n$, $999675 = 5^{344}\cdot10245^n$. 

Look up $5^1 \mod p$, $5^3 \mod p$, etc. in the table of targets.

$5^{755} \mod p = 966603$; find $966603 = 5^{332}\cdot10245^n$ in the table of targets, so $755 = 332\cdot1024 + n \mod p$; deduce $n = 660789$. 

...
Faster: Increase 100 to $\approx \sqrt{p}$.
Only $\approx 2\sqrt{p}$ mults
to solve one DL problem!

“Shanks baby-step-giant-step
discrete-logarithm algorithm.”

Example: $p = 1000003$,
$5^n \mod p = 262682$.

Compute $5^{1024} \mod p = 58588$.
Then compute 1000 targets:
$5^{1024}5^n \mod p = 966849$,
$5^{2\cdot1024}5^n \mod p = 579277$,
$5^{3\cdot1024}5^n \mod p = 579062$, ...,
$5^{1000\cdot1024}5^n \mod p = 321705$.

Build a sorted table of targets:
$2573 = 5^{430\cdot1024}5^n \mod p$,
$3371 = 5^{192\cdot1024}5^n \mod p$,
$3593 = 5^{626\cdot1024}5^n \mod p$,
$4960 = 5^{663\cdot1024}5^n \mod p$,
$5218 = 5^{376\cdot1024}5^n \mod p$, ...
$999675 = 5^{344\cdot1024}5^n \mod p$.

Look up $5^1 \mod p$, $5^2 \mod p$, $5^3 \mod p$, etc. in this table.
$5^{755} \mod p = 966603$; find
$966603 = 5^{332\cdot1024}5^n \mod p$
in the table of targets;
so $755 = 332 \cdot 1024 + n \mod p$;
deduce $n = 660789$. 

Faster: Increase 100 to $\approx \sqrt{p}$. Only $\approx 2\sqrt{p}$ mults to solve one DL problem!

“Shanks baby-step-giant-step discrete-logarithm algorithm.”

Example: $p = 1000003$, $5^n \mod p = 262682$.

Compute $5^{1024} \mod p = 58588$.

Then compute 1000 targets:

$5^{1024}5^n \mod p = 966849$,

$5^2\cdot10245^n \mod p = 579277$,

$5^3\cdot10245^n \mod p = 579062$, \ldots,

$5^{1000}\cdot10245^n \mod p = 321705$.

Build a sorted table of targets:

$2573 = 5^{430\cdot1024}5^n \mod p$,

$3371 = 5^{192\cdot1024}5^n \mod p$,

$3593 = 5^{626\cdot1024}5^n \mod p$,

$4960 = 5^{663\cdot1024}5^n \mod p$,

$5218 = 5^{376\cdot1024}5^n \mod p$, \ldots,

$999675 = 5^{344\cdot1024}5^n \mod p$.

Look up $5^1 \mod p$, $5^2 \mod p$, $5^3 \mod p$, etc. in this table.

$5^{755} \mod p = 966603$; find $966603 = 5^{332\cdot1024}5^n \mod p$ in the table of targets; so $755 = 332\cdot1024 + n \mod p - 1$; deduce $n = 660789$. 
Increase 100 to $\approx \sqrt{p}$.

2 $\sqrt{p}$ mults to solve one DL problem!

"Shanks baby-step-giant-step logarithm algorithm."

Example: $p = 1000003$.

Compute $5^{1024} \mod p = 262682$.

Build a sorted table of targets:

- $2573 = 5^{430 \cdot 1024} \cdot 5^n \mod p$,
- $3371 = 5^{192 \cdot 1024} \cdot 5^n \mod p$,
- $3593 = 5^{626 \cdot 1024} \cdot 5^n \mod p$,
- $4960 = 5^{663 \cdot 1024} \cdot 5^n \mod p$,
- $5218 = 5^{376 \cdot 1024} \cdot 5^n \mod p$,

Eliminating storage

Improved method: Define $x_0 = 1$; $x_{i+1} = 5 \cdot x_i \mod p$ if $x_i < 3 \cdot Z$;

$x_{i+1} = x_{2i} \mod p$ if $x_i < 2^2 + 3 \cdot Z$;

$x_{i+1} = 5 \cdot n \cdot x_i \mod p$ otherwise.

Search for a collision in $x_i$:

$x_1 = x_2 \iff x_2 = x_4 \iff x_3 = x_6 \iff x_4 = x_8 \iff \ldots$

Deduce linear equation for $n$. 

Look up $5^1 \mod p$, $5^2 \mod p$, $5^3 \mod p$, etc. in this table.

Compute 1000 targets:

- $2573 \mod p = 58588$,
- $3371 \mod p = 579277$,
- $3593 \mod p = 579062$, …,
- $999675 \mod p = 321705$.

Now: $p = 1000003$,

$p = 262682$.

Find $5^{1024} \mod p = 58588$.

Compute 1000 targets:

- $2573 \mod p = 58588$,
- $3371 \mod p = 5663 \cdot 1024 \cdot 5^n \mod p$,
- $3593 \mod p = 579277$,
- $4960 \mod p = 579062$, …,
- $999675 \mod p = 321705$.

Look up $5^1 \mod p$, $5^2 \mod p$, $5^3 \mod p$, etc. in this table.

Find $5^{755} \mod p = 966603$; find

$966603 = 5^{332 \cdot 1024} \cdot 5^n \mod p$ in the table of targets;

so $755 = 332 \cdot 1024 + n \mod p - 1$; deduce $n = 660789$.
Faster: Increase 100 to \( \sqrt{p} \).

Only 2 mults to solve one DL problem!


Example:

\[ p = 1000003, \]

\[ 5^{1024} \mod p = 262682. \]

Compute \( 5^{1024} \mod p = 58588. \)

Then compute 1000 targets:

\[ 5^{1024} 5^{n} \mod p = 966849, \]
\[ 5^{1024} 5^{2n} \mod p = 579277, \]
\[ 5^{1024} 5^{3n} \mod p = 579062, \ldots, \]
\[ 5^{1024} 5^{1000n} \mod p = 321705. \]

Eliminating storage

Improved method: Define \( x_0 = 1; \)
\[ x_{i+1} = 5x_i \mod p \]
\[ x_{i+1} = x_i^2 \mod p \]
\[ x_{i+1} = 5^n x_i \mod p \]

Then \( x_i = 5^{a_i n + b_i} \mod p \)
where \( (a_0, b_0) = (0, 0) \)
\( (a_{i+1}, b_{i+1}) = (a_i, b_i + 1) \)
\( (a_{i+1}, b_{i+1}) = (2a_i, 2b_i) \)
\( (a_{i+1}, b_{i+1}) = (a_i + 1, b_i) \)

Search for a collision in \( x_i : \)
\[ x_1 = x_2 ? \]
\[ x_2 = x_4 ? \]
\[ x_4 = x_8 ? \]
Deduce linear equation for \( n \).
Faster: Increase 100 to $p$.

Only $2p$ mults to solve one DL problem!


Example: $p = 1000003$, $5^n \mod p = 262682$.

Compute $5^{1024} \mod p = 58588$.

Then compute 1000 targets:

$5^{1024} \times 5^n \mod p = 966849$, $5^{2 \times 1024} \mod p = 579277$, $5^{3 \times 1024} \mod p = 579062$, ... , $5^{1000 \times 1024} \mod p = 321705$.

Build a sorted table of targets:

$2573 = 5^{430 \cdot 1024} 5^n \mod p$, $3371 = 5^{192 \cdot 1024} 5^n \mod p$, $3593 = 5^{626 \cdot 1024} 5^n \mod p$, $4960 = 5^{663 \cdot 1024} 5^n \mod p$, $5218 = 5^{376 \cdot 1024} 5^n \mod p$, ... , $999675 = 5^{344 \cdot 1024} 5^n \mod p$.

Look up $5^1 \mod p$, $5^2 \mod p$, $5^3 \mod p$, etc. in this table.

$5^{755} \mod p = 966603$; find $966603 = 5^{332 \cdot 1024} 5^n \mod p$ in the table of targets; so $755 = 332 \cdot 1024 + n \mod p - 1$; deduce $n = 660789$.

Eliminating storage

Improved method: Define $x_0 = 1$;

$x_{i+1} = 5^n x_i \mod p$ if $x_i \in 32$;

$x_{i+1} = x_i^2 \mod p$ if $x_i \in 2 + 32$;

$x_{i+1} = 5^n x_i \mod p$ otherwise.

Then $x_i = 5^{a_i n + b_i} \mod p$ where $(a_0, b_0) = (0, 0)$ and

$(a_{i+1}, b_{i+1}) = (a_i, b_i + 1)$, or

$(a_{i+1}, b_{i+1}) = (2a_i, 2b_i)$, or

$(a_{i+1}, b_{i+1}) = (a_i + 1, b_i)$.

Search for a collision in $x_i$:

$x_1 = x_2$? $x_2 = x_4$? $x_3 = x_6$? $x_4 = x_8$? $x_5 = x_{10}$? etc.

Deduce linear equation for $n$.
### Build a sorted table of targets:

<table>
<thead>
<tr>
<th>Number</th>
<th>Modulo Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2573</td>
<td>$5^{430} \cdot 1024 \cdot 5^n \mod p$</td>
</tr>
<tr>
<td>3371</td>
<td>$5^{192} \cdot 1024 \cdot 5^n \mod p$</td>
</tr>
<tr>
<td>3593</td>
<td>$5^{626} \cdot 1024 \cdot 5^n \mod p$</td>
</tr>
<tr>
<td>4960</td>
<td>$5^{663} \cdot 1024 \cdot 5^n \mod p$</td>
</tr>
<tr>
<td>5218</td>
<td>$5^{376} \cdot 1024 \cdot 5^n \mod p$</td>
</tr>
<tr>
<td>999675</td>
<td>$5^{344} \cdot 1024 \cdot 5^n \mod p$</td>
</tr>
</tbody>
</table>

### Eliminating storage

#### Improved method:

Define $x_0 = 1$; $x_{i+1} = 5^n x_i \mod p$ if $x_i \in 3 \mathbb{Z}$; $x_{i+1} = x_i^2 \mod p$ if $x_i \in 2 + 3 \mathbb{Z}$; $x_{i+1} = 5^n x_i \mod p$ otherwise.

Then $x_i = 5^{a_i n + b_i} \mod p$ where $(a_0, b_0) = (0, 0)$ and

- $(a_{i+1}, b_{i+1}) = (a_i, b_i + 1)$, or
- $(a_{i+1}, b_{i+1}) = (2a_i, 2b_i)$, or
- $(a_{i+1}, b_{i+1}) = (a_i + 1, b_i)$.

Search for a collision in $x_i$: $x_1 = x_2$? $x_2 = x_4$? $x_3 = x_6$? $x_4 = x_8$? $x_5 = x_{10}$? etc.

Deduce linear equation for $n$. 

---

Look up $5^1 \mod p$, $5^2 \mod p$, $5^3 \mod p$, etc. in this table.

$5^{755} \mod p = 966603$; find $966603 = 5^{332} \cdot 1024 \cdot 5^n \mod p$ in the table of targets; so $755 = 332 \cdot 1024 + n \mod p - 1$; deduce $n = 660789$. 

Deduce linear equation for $n$. 

---
Eliminating storage

Improved method: Define $x_0 = 1$;
$x_{i+1} = 5x_i \mod p$ if $x_i \in 3\mathbb{Z}$;
$x_{i+1} = x_i^2 \mod p$ if $x_i \in 2 + 3\mathbb{Z}$;
$x_{i+1} = 5^n x_i \mod p$ otherwise.

Then $x_i = 5^{a_i n + b_i} \mod p$
where $(a_0, b_0) = (0, 0)$ and
$(a_{i+1}, b_{i+1}) = (a_i, b_i + 1)$, or
$(a_{i+1}, b_{i+1}) = (2a_i, 2b_i)$, or
$(a_{i+1}, b_{i+1}) = (a_i + 1, b_i)$.

Search for a collision in $x_i$:
$x_1 = x_2? \ x_2 = x_4? \ x_3 = x_6? \ x_4 = x_8? \ x_5 = x_{10}? \ etc.$

Deduce linear equation for $n$. 

The $x_i$'s enter a cycle, typically within $p$ steps.

Example: 1000003, 262682.
Modulo 1000003:
$x_1 = 5^n \mod p$
$x_2 = 5^{2n} \mod p$
$x_3 = 5^{3n} \mod p$
$x_4 = 5^{4n} \mod p$
$x_5 = 5^{5n} \mod p$
$x_6 = 5^{6n} \mod p$
$x_7 = 5^{7n} \mod p$
$x_8 = 5^{8n} \mod p$

etc.
Example of targets:
\[ n \mod p,\]
\[ n \mod p,\]
\[ n \mod p,\]
\[ n \mod p, \ldots,\]
\[ 5^n \mod p.\]

Look up \(5^n\mod p, 5^2 \mod p, \ldots\) in this table.

Eliminating storage
Improved method: Define \(x_0 = 1;\)
\[ x_{i+1} = 5x_i \mod p \quad \text{if} \quad x_i \in 3\mathbb{Z};\]
\[ x_{i+1} = x_i^2 \mod p \quad \text{if} \quad x_i \in 2 + 3\mathbb{Z};\]
\[ x_{i+1} = 5^n x_i \mod p \quad \text{otherwise.}\]

Then \(x_i = 5^{a_i n + b_i} \mod p\)
where \((a_0, b_0) = (0, 0)\) and
\((a_{i+1}, b_{i+1}) = (a_i, b_i + 1),\)
\((a_{i+1}, b_{i+1}) = (2a_i, 2b_i),\)
\((a_{i+1}, b_{i+1}) = (a_i + 1, b_i).\)

Search for a collision in \(x_i:\)
\[ x_1 = x_2? \quad x_2 = x_4? \quad x_3 = x_6? \]
\[ x_4 = x_8? \quad x_5 = x_{10}? \quad \text{etc.}\]
Deduce linear equation for \(n.\)

The \(x_i\)'s enter a cycle, typically within \(\approx p\) steps.

Example: 1000003, 262682.

Modulo 1000003:
\[ x_1 = 5^n = 262682;\]
\[ x_2 = 5^{2n} = 262682;\]
\[ x_3 = 5^{2n+1} = 626121;\]
\[ x_4 = 5^{2n+2} = 130596;\]
\[ x_5 = 5^{2n+3} = 652980;\]
\[ x_6 = 5^{2n+4} = 264891;\]
\[ x_7 = 5^{4n+8} = 324452;\]
\[ x_8 = 5^{4n+9} = 324452;\]
\[ \text{etc.}\]
Eliminating storage

Improved method: Define $x_0 = 1$;
$x_{i+1} = 5x_i \mod p$ if $x_i \in 3\mathbb{Z}$;
$x_{i+1} = x_i^2 \mod p$ if $x_i \in 2 + 3\mathbb{Z}$;
$x_{i+1} = 5^n x_i \mod p$ otherwise.

Then $x_i = 5^{a_i n + b_i} \mod p$
where $(a_0, b_0) = (0, 0)$ and
$(a_{i+1}, b_{i+1}) = (a_i, b_i + 1)$, or
$(a_{i+1}, b_{i+1}) = (2a_i, 2b_i)$, or
$(a_{i+1}, b_{i+1}) = (a_i + 1, b_i)$.

Search for a collision in $x_i$:
$x_1 = x_2? \ x_2 = x_4? \ x_3 = x_6? \ x_4 = x_8? \ x_5 = x_{10}? \ etc.$

Deduce linear equation for $n$.

The $x_i$'s enter a cycle, typically within $\approx \sqrt{p}$ steps.

Example: 1000003, 262682.

Modulo 1000003:
$x_1 = 5^n = 262682$.
$x_2 = 5^{2n} = 262682^2 = 626121$.
$x_3 = 5^{2n+1} = 5 \cdot 626121 = 130596$.
$x_4 = 5^{2n+2} = 5 \cdot 130596 = 652980$.
$x_5 = 5^{2n+3} = 5 \cdot 652980 = 264891$.
$x_6 = 5^{2n+4} = 5 \cdot 264891 = 324452$.
$x_7 = 5^{4n+8} = 324452^2 = 784500$.
$x_8 = 5^{4n+9} = 5 \cdot 784500 = 922491$.
etc.
Eliminating storage

Improved method: Define $x_0 = 1$;

$x_{i+1} = 5x_i \mod p$ if $x_i \in 3\mathbb{Z}$;

$x_{i+1} = x_i^2 \mod p$ if $x_i \in 2 + 3\mathbb{Z}$;

$x_{i+1} = 5^n x_i \mod p$ otherwise.

Then $x_i = 5^{a_in+b_i} \mod p$

where $(a_0, b_0) = (0, 0)$ and

$(a_{i+1}, b_{i+1}) = (a_i, b_i + 1)$, or

$(a_{i+1}, b_{i+1}) = (2a_i, 2b_i)$, or

$(a_{i+1}, b_{i+1}) = (a_i + 1, b_i)$.

Search for a collision in $x_i$:

$x_1 = x_2$? $x_2 = x_4$? $x_3 = x_6$?

$x_4 = x_8$? $x_5 = x_{10}$? etc.

Deduce linear equation for $n$.

The $x_i$’s enter a cycle, typically within $\approx \sqrt{p}$ steps.

Example: 1000003, 262682.

Modulo 1000003:

$x_1 = 5^n = 262682$.

$x_2 = 5^{2n} = 262682^2 = 626121$.

$x_3 = 5^{2n+1} = 5 \cdot 626121 = 130596$.

$x_4 = 5^{2n+2} = 5 \cdot 130596 = 652980$.

$x_5 = 5^{2n+3} = 5 \cdot 652980 = 264891$.

$x_6 = 5^{2n+4} = 5 \cdot 264891 = 324452$.

$x_7 = 5^{4n+8} = 324452^2 = 784500$.

$x_8 = 5^{4n+9} = 5 \cdot 784500 = 922491$. etc.
Improving storage

Improved method: Define $x_0 = 1$;

- $5x_i \mod p$ if $x_i \in 3\mathbb{Z}$;
- $x_i^2 \mod p$ if $x_i \in 2 + 3\mathbb{Z}$;
- $5^n x_i \mod p$ otherwise.

Deduce linear equation for $n$.

The $x_i$'s enter a cycle, typically within $\approx \sqrt{p}$ steps.

Example: 1000003, 262682.

Modulo 1000003:

- $x_1 = 5^n = 262682$.
- $x_2 = 5^{2n} = 262682^2 = 626121$.
- $x_3 = 5^{2n+1} = 5 \cdot 626121 = 3130596$.
- $x_4 = 5^{2n+2} = 5 \cdot 130596 = 652980$.
- $x_5 = 5^{2n+3} = 5 \cdot 652980 = 264891$.
- $x_6 = 5^{2n+4} = 5 \cdot 264891 = 324452$.
- $x_7 = 5^{4n+8} = 324452^2 = 784500$.
- $x_8 = 5^{4n+9} = 5 \cdot 784500 = 922491$.

etc.

$x_1785 = \cdots$  
$x_3570 = \cdots$

(Cycle length is 357.)

Conclude that $249847 n + 759123 \equiv 555013 \mod (p-1)$,  
$388795 n + 632781 \equiv 555013 \mod (p-1)$,  
so $n \equiv 160788 \mod (p-1)$.

Only 6 possible $n$'s.

Try each of them.

Find that $5^n \mod p = 262682$  
for $n = 160788 + 3(p-1) = 6$, i.e.,  
for $n = 660789$. 

The $x_i$'s enter a cycle, typically within $\approx \sqrt{p}$ steps. 

Example: 1000003, 262682. 

Modulo 1000003: 

- $x_1 = 5^n = 262682$. 
- $x_2 = 5^{2n} = 262682^2 = 626121$. 
- $x_3 = 5^{2n+1} = 5 \cdot 626121 = 3130596$. 
- $x_4 = 5^{2n+2} = 5 \cdot 130596 = 652980$. 
- $x_5 = 5^{2n+3} = 5 \cdot 652980 = 264891$. 
- $x_6 = 5^{2n+4} = 5 \cdot 264891 = 324452$. 
- $x_7 = 5^{4n+8} = 324452^2 = 784500$. 
- $x_8 = 5^{4n+9} = 5 \cdot 784500 = 922491$. 

etc.
Define $x_0 = 1$; if $x_i \in 3 \mathbb{Z}$; if $x_i \in 2 + 3 \mathbb{Z}$; $p$ otherwise.

The $x_i$'s enter a cycle, typically within $\approx \sqrt{p}$ steps.

Example: 1000003, 262682.

Modulo 1000003:

\[
x_1 = 5^n = 262682.
\]

\[
x_2 = 5^{2n} = 262682^2 = 626121.
\]

\[
x_3 = 5^{2n+1} = 5 \cdot 626121 = 130596.
\]

\[
x_4 = 5^{2n+2} = 5 \cdot 130596 = 652980.
\]

\[
x_5 = 5^{2n+3} = 5 \cdot 652980 = 264891.
\]

\[
x_6 = 5^{2n+4} = 5 \cdot 264891 = 324452.
\]

\[
x_7 = 5^{4n+8} = 324452^2 = 784500.
\]

\[
x_8 = 5^{4n+9} = 5 \cdot 784500 = 922491.
\]

\[\text{etc.}\]

Conclude that $249847n + 759123 \equiv 0 \pmod{1000003}$, so $n \equiv 160788 \pmod{\frac{1000003-1}{2}}$.

Only 6 possible $n$'s.

Try each of them.

Find that $5^n \mod 1000003 = 262682$ for $n = 160788 + 3(1000003 - 1) = 6$; i.e., for $n = 660789$. 

(Cycle length is 357.)

\[
x_{1785} = 5^{249847n+759123} \equiv 784500 \pmod{1000003}.
\]

\[
x_{3570} = 5^{388795n+632781} \equiv 264891 \pmod{1000003}.
\]
Eliminating storage

Improved method: Define \( x_0 = 1; \)
\( x_{i+1} = 5x_i \mod p \) if \( x_i \not\in \mathbb{Z}; \)
\( x_{i+1} = x_i^2 \mod p \) otherwise.

Then \( x_i = 5^{a_i n + b_i} \mod p \) where \((a_0, b_0) = (0, 0)\) and \((a_{i+1}, b_{i+1}) = (a_i, b_i + 1), \) or \((a_{i+1}, b_{i+1}) = (2a_i, 2b_i), \) or \((a_{i+1}, b_{i+1}) = (a_i + 1, b_i).\)

Search for a collision in \( x_i: \)
\( x_1 = x_2 = x_4 = x_6 = \ldots \)
\( x_4 = x_8 = x_{10} = \ldots \)

Deduce linear equation for \( n. \)

The \( x_i's \) enter a cycle, typically within \( \approx \sqrt{p} \) steps.

Example: 1000003, 262682.

Modulo 1000003:
\( x_1 = 5^n = 262682. \)
\( x_2 = 5^{2n} = 262682^2 = 626121. \)
\( x_3 = 5^{2n+1} = 5 \cdot 626121 = 130596. \)
\( x_4 = 5^{2n+2} = 5 \cdot 130596 = 652980. \)
\( x_5 = 5^{2n+3} = 5 \cdot 652980 = 264891. \)
\( x_6 = 5^{2n+4} = 5 \cdot 264891 = 324452. \)
\( x_7 = 5^{4n+8} = 324452^2 = 784500. \)
\( x_8 = 5^{4n+9} = 5 \cdot 784500 = 922491. \)

(Cycle length is 357.)

Conclude that
\( 249847n + 759123 \equiv 388795n + 632781 \pmod{p}, \)
so \( n \equiv 160788 \pmod{(p - 1)/6}. \)

Only 6 possible \( n \)'s.

Try each of them.

Find that \( 5^n \pmod{p} = 262682 \)
for \( n = 160788 + 3(p - 1)/6, \)
for \( n = 660789. \)
The \( x_i \)'s enter a cycle, typically within \( \sim \sqrt{\rho} \) steps.

Example: 1000003, 262682.

Modulo 1000003:
\[
x_1 = 5^n = 262682.
\]
\[
x_2 = 5^{2n} = 262682^2 = 626121.
\]
\[
x_3 = 5^{2n+1} = 5 \cdot 626121 = 130596.
\]
\[
x_4 = 5^{2n+2} = 5 \cdot 130596 = 652980.
\]
\[
x_5 = 5^{2n+3} = 5 \cdot 652980 = 264891.
\]
\[
x_6 = 5^{2n+4} = 5 \cdot 264891 = 324452.
\]
\[
x_7 = 5^{4n+8} = 324452^2 = 784500.
\]
\[
x_8 = 5^{4n+9} = 5 \cdot 784500 = 922491.
\]

etc.

\[
x_{1785} = 5^{249847n+759123} = 555013.
\]
\[
x_{3570} = 5^{388795n+632781} = 555013.
\]

(Cycle length is 357.)

Conclude that
\[
249847n + 759123 \equiv 388795n + 632781 \pmod{p - 1},
\]
so \( n \equiv 160788 \pmod{(p - 1)/6} \).

Only 6 possible \( n \)'s.

Try each of them.

Find that \( 5^n \bmod p = 262682 \)
for \( n = 160788 + 3(p - 1)/6 \), i.e.,
for \( n = 660789 \).
The $x_i$'s enter a cycle, typically within $\approx \sqrt{p}$ steps.

Example: 1000003, 262682.

1000003:

$x_1 = 5$

$x_2 = 5^2 = 262682$.

$x_3 = 5^{2^2} = 130596$.

$x_4 = 5^{2^3} = 652980$.

$x_5 = 5^{2^4} = 264891$.

$x_6 = 5^{2^5} = 324452$.

$x_7 = 5^{2^8} = 784500$.

$x_8 = 5^{2^9} = 922491$.

(Cycle length is 357.)

Conclude that

$x_{1785} = 5^{249847n + 759123} = 555013$.

$x_{3570} = 5^{388795n + 632781} = 555013$.

This is "Pollard's rho method."

Optimized:

Another method, similar speed:

"Pollard's kangaroo method."

Can parallelize both methods.

"van Oorschot/Wiener parallel DL using distinguished points."

Bottom line: With $c$ mults, distributed across many cores, have chance of finding $n$ from $5^n \mod p$.

With $2^{90}$ mults (a few years?), have chance of $2^{180} = p$.

Negligible if, e.g., $p \approx 2^{256}$. 

The $x_i$'s enter a cycle, typically within $p$ steps.

Example: $1000003, 262682$.

Modulo $1000003$:

$x_1 = 5$

$n = 262682$

$x_2 = 5^2n = 626121$.

$x_3 = 5^{2n+1} = 130596$.

$x_4 = 5^{2n+2} = 652980$.

$x_5 = 5^{2n+3} = 264891$.

$x_6 = 5^{2n+4} = 324452$.

$x_7 = 5^{4n+8} = 784500$.

$x_8 = 5^{4n+9} = 922491$.

(Cycle length is 357.)

Conclude that

$249847n + 759123 \equiv 388795n + 632781 \pmod{p - 1}$,

so $n \equiv 160788 \pmod{(p - 1)/6}$.

Only 6 possible $n$'s.

Try each of them.

Find that $5^n \pmod{p} = 262682$

for $n = 160788 + 3(p - 1)/6$, i.e.,

for $n = 660789$.

This is “Pollard’s rho method.”

Optimized: $\approx \sqrt{p}$

Another method, similar speed:

“Pollard’s kangaroo method.”

Can parallelize both methods:

“van Oorschot/Wiener parallel DL using distinguished points.”

Bottom line: With $c$ mults, distributed across many cores,

have chance $\approx c^2 / p$ of finding $n$ from $5^n \pmod{p}$.

With $2^{90}$ mults (a few years?),

have chance $\approx 2^{18}$.

Negligible if, e.g., $p = 2^{256}$. 

The $x_i$'s enter a cycle, typically within $p$ steps. Example: 1000003, 262682. Modulo 1000003:

$x_1 = 5$

$n = 262682$.

$x_2 = 5^2 n = 626121$.

$x_3 = 5^{2+1} n = 130596$.

$x_4 = 5^{2+2} n = 652980$.

$x_5 = 5^{2+3} n = 264891$.

$x_6 = 5^{2+4} n = 324452$.

$x_7 = 5^{4+8} n = 784500$.

$x_8 = 5^{4+9} n = 922491$.

etc.

(Cycle length is 357.)

Conclude that

$249847n + 759123 \equiv 262682 \pmod{p}$, so $n \equiv 160788 \pmod{(p - 1)/6}$.

Only 6 possible $n$'s.

Try each of them.

Find that $5^n \pmod{p} = 262682$ for $n = 160788 + 3(p - 1)/6$, i.e., for $n = 660789$.

This is "Pollard’s rho method." Optimized: $\approx \sqrt{p}$ mults.

Another method, similar speed: "Pollard’s kangaroo method." Can parallelize both methods.

"van Oorschot/Wiener parallel DL using distinguished points."

Bottom line: With $c$ mults, distributed across many cores, have chance $\approx c^2 / p$ of finding $n$ from $5^n \pmod{p}$.

With $2^{90}$ mults (a few years?), have chance $\approx 2^{180} / p$.

Negligible if, e.g., $p \approx 2^{256}$. 

$x_{1785} = 5^{249847n + 759123} = 555013.$

$x_{3570} = 5^{388795n + 632781} = 555013.$

This is “Pollard’s rho method.” Optimized: $\approx \sqrt{p}$ mults. Another method, similar speed: “Pollard’s kangaroo method.” Can parallelize both methods. “van Oorschot/Wiener parallel DL using distinguished points.” Bottom line: With $c$ mults, distributed across many cores, have chance $\approx c^2 / p$ of finding $n$ from $5^n \pmod{p}$. With $2^{90}$ mults (a few years?), have chance $\approx 2^{180} / p$. Negligible if, e.g., $p \approx 2^{256}$. 

$121.$

$30596.$

$952980.$

$164891.$

$324500.$

$822491.$
\[x_{1785} = 5^{249847n+759123} = 555013.\]
\[x_{3570} = 5^{388795n+632781} = 555013.\]

(Cycle length is 357.)

Conclude that
\[249847n + 759123 \equiv 388795n + 632781 \pmod{p-1},\]
so \( n \equiv 160788 \pmod{(p-1)/6}.\)

Only 6 possible \( n \)'s.
Try each of them.
Find that \( 5^n \mod p = 262682 \)
for \( n = 160788 + 3(p-1)/6, \) i.e.,
for \( n = 660789.\)

This is “Pollard’s rho method.”
Optimized: \( \approx \sqrt{p} \) mults.
Another method, similar speed:
“Pollard’s kangaroo method.”

Can parallelize both methods.
“van Oorschot/Wiener parallel DL using distinguished points.”

Bottom line: With \( c \) mults,
distributed across many cores,
have chance \( \approx c^2/p \)
of finding \( n \) from \( 5^n \mod p.\)

With \( 2^{90} \) mults (a few years?),
have chance \( \approx 2^{180}/p.\)
Negligible if, e.g., \( p \approx 2^{256}.\)
This is “Pollard’s rho method.”
Optimized: $\approx \sqrt{p}$ mults.
Another method, similar speed: “Pollard’s kangaroo method.”

Can parallelize both methods.
“van Oorschot/Wiener parallel DL using distinguished points.”

Bottom line: With $c$ mults, distributed across many cores, have chance $\approx c^2/p$ of finding $n$ from $5^n \mod p$.

With $2^{90}$ mults (a few years?), have chance $\approx 2^{180}/p$.
Negligible if, e.g., $p \approx 2^{256}$.

Factors of the group order
Assume 5 has order $ab$.

Given $x$, a power of 5:
$5^a$ has order $b$, and $x^a$ is a power of $5^b$.
Compute $m = \log_5 (x = 5^b)$.
Then $x = 5^b + mb$.

$5^b$ has order $a$, and $x/5^l$ is a power of $5^b$.
Compute $m = \log_{5^b} (x/5^l)$.
Then $x = 5^b + mb$.
This is “Pollard’s rho method.” Optimized: $\approx \sqrt{p}$ mults.
Another method, similar speed: “Pollard’s kangaroo method.”

Can parallelize both methods. “van Oorschot/Wiener parallel DL using distinguished points.”

Bottom line: With $c$ mults, distributed across many cores, have chance $\approx c^2/p$ of finding $n$ from $5^n \mod p$.

With $2^{90}$ mults (a few years?), have chance $\approx 2^{180}/p$.
Negligible if, e.g., $p \approx 2^{256}$.

Factors of the group order
Assume 5 has order $ab$.
Given $x$, a power of 5:
$5^a$ has order $b$, and $x^a$ is a power of $5^a$.
Compute $l = \log_{5^a} x$.
$5^b$ has order $a$, and $x/5^l$ is a power of $5^b$.
Compute $m = \log_{5^b} (x/5^l)$.
Then $x = 5^{l+mb}$.

Assume $5 \equiv 1 \pmod{p-1}$, $3 (p-1)/6$, i.e., $p = 262682$
Factors of the group order
Assume 5 has order \( ab \).

Given \( x \), a power of 5:

\[
5^a \text{ has order } b, \text{ and } x^a \text{ is a power of } 5^a.
\]

Compute \( \ell = \log_{5^a} x^a \).

\[
5^b \text{ has order } a, \text{ and } x/5^\ell \text{ is a power of } 5^b.
\]

Compute \( m = \log_{5^b}(x/5^\ell) \).

Then \( x = 5^\ell + mb \).

This is “Pollard's rho method.”
Optimized: \( \approx \sqrt{p} \) mults.

Another method, similar speed:

“Pollard's kangaroo method.”

Can parallelize both methods.

“van Oorschot/Wiener parallel DL using distinguished points.”

Bottom line: With \( c \) mults, distributed across many cores, have chance \( \approx c^2/p \)
of finding \( n \) from \( 5^n \mod p \).

With \( 2^{90} \) mults (a few years?), have chance \( \approx 2^{180}/p \).

Negligible if, e.g., \( p \approx 2^{256} \).
This is “Pollard’s rho method.”
Optimized: \( \approx \sqrt{p} \) mults.
Another method, similar speed:
“Pollard’s kangaroo method.”
Can parallelize both methods.
“van Oorschot/Wiener parallel DL using distinguished points.”

Bottom line: With \( c \) mults, distributed across many cores, have chance \( \approx c^2/p \) of finding \( n \) from \( 5^n \mod p \).

With \( 2^{90} \) mults (a few years?), have chance \( \approx 2^{180}/p \).
Negligible if, e.g., \( p \approx 2^{256} \).

Factors of the group order
Assume 5 has order \( ab \).
Given \( x \), a power of 5:
\( 5^a \) has order \( b \), and \( x^a \) is a power of \( 5^a \).
Compute \( \ell = \log_{5^a} x^a \).
\( 5^b \) has order \( a \), and \( x/5^\ell \) is a power of \( 5^b \).
Compute \( m = \log_{5^b} (x/5^\ell) \).
Then \( x = 5^{\ell+mb} \).
“Pollard’s rho method.”

Optimized: \( \approx \sqrt{p} \) mults.

Another method, similar speed:

“Pollard’s kangaroo method.”

Can parallelize both methods.

van Oorschot/Wiener parallel DL using distinguished points.”

Bottom line: With \( c^{2} \) mults,
distributed across many cores,
have chance \( c^{2} / p \)
of finding \( n \) from \( 5^{n} \) mod \( p \).

With \( 2^{90} \) mults (a few years?),
have chance \( 2^{180} / p \).

Negligible if, e.g., \( p = 2^{256} \).

Factors of the group order

Assume 5 has order \( ab \).

Given \( x \), a power of 5:

\( 5^{a} \) has order \( b \), and
\( x^{a} \) is a power of \( 5^{a} \).

Compute \( \ell = \log_{5^{a}} x^{a} \).

\( 5^{b} \) has order \( a \), and
\( x/5^{\ell} \) is a power of \( 5^{b} \).

Compute \( m = \log_{5^{b}} (x/5^{\ell}) \).

Then \( x = 5^{\ell + mb} \).

This “Pohlig-Hellman method” converts an order- \( ab \) DL into an order- \( a \) DL, an order- \( b \) DL,

and a few exponentiations.

e.g. \( p = 1000003 \), \( x = 262682 \):

\( p - 1 = 160787 \times 6101 \).

Compute \( \log_{5} 160787 \).

Compute \( \log_{5} 6101 \).

Then \( x = 1000002 \).

Use rho:

Better if \( ab \) factors further:
avoid Pohlig-Hellman recursively.
This is “Pollard’s rho method.”

Optimized:

\[ p \] mults.

Another method, similar speed:

“Pollard’s kangaroo method.”

Can parallelize both methods.

“van Oorschot/Wiener parallel DL using distinguished points.”

Bottom line: With \( c \) mults, distributed across many cores, have chance \( c^2 = p \) of finding \( n \) from \( 5^n \mod p \).

With \( 2^{90} \) mults (a few years?), have chance \( 2^{180} = p \).

Negligible if, e.g., \( p \approx 2^{256} \).

This “Pohlig-Hellman method” converts an order-\( ab \) DL into an order-\( a \) DL, an order-\( b \) DL, and a few exponentiations.

e.g. \( p = 1000003 \), \( x = 262682 \):

\( p - 1 = 6b \) where \( b = 166667 \).

Compute \( \log_{5^6} (x^6) = 160788 \).

Compute \( x = 5^{160788} = 1000002 \).

Compute \( \log_{5^b} 1000002 = 3 \).

Then \( x = 5^{160788 + 3b} = 5^{660789} \).

Use rho: \( \approx \sqrt{a} + \sqrt{b} \).

Better if \( ab \) factors further:

apply Pohlig-Hellman recursively.

Factors of the group order

Assume 5 has order \( ab \).

Given \( x \), a power of 5:

\( 5^a \) has order \( b \), and \( x^a \) is a power of \( 5^a \).

Compute \( \ell = \log_{5^a} x^a \).

\( 5^b \) has order \( a \), and \( x/5^\ell \) is a power of \( 5^b \).

Compute \( m = \log_{5^b} (x/5^\ell) \).

Then \( x = 5^{\ell + mb} \).
Factors of the group order

Assume 5 has order \( ab \).

Given \( x \), a power of 5:

5\(^a\) has order \( b \), and 
\( x^a \) is a power of 5\(^a\).

Compute \( \ell = \log_{5^a} x^a \).

5\(^b\) has order \( a \), and 
\( x/5^{\ell} \) is a power of 5\(^b\).

Compute \( m = \log_{5^b} (x/5^{\ell}) \).

Then \( x = 5^{\ell+m}b \).

This “Pohlig-Hellman method” converts an order-\( ab \) DL into an order-\( a \) DL, an order-\( b \) DL, and a few exponentiations.

e.g. \( p = 1000003 \), \( x = 262682 \):

\( p - 1 = 6b \) where \( b = 166667 \).

Compute \( \log_{5^6} (x^6) = 160788 \).

Compute \( x/5^{160788} = 1000002 \).

Compute \( \log_{5^b} 1000002 = 3 \).

Then \( x = 5^{160788+3b} = 5^{660789} \).

Use rho: \( \approx \sqrt{a} + \sqrt{b} \) mults.

Better if \( ab \) factors further:

apply Pohlig-Hellman recursively.
Factors of the group order

Assume 5 has order $ab$.

Given $x$, a power of 5:

$5^a$ has order $b$, and
$x^a$ is a power of $5^a$.
Compute $l = \log_{5^a} x^a$.

$5^b$ has order $a$, and
$x/5^l$ is a power of $5^b$.
Compute $m = \log_{5^b}(x/5^l)$.

Then $x = 5^l + mb$.

This “Pohlig-Hellman method” converts an order-\(ab\) DL into an order-\(a\) DL, an order-\(b\) DL, and a few exponentiations.

e.g. $p = 1000003$, $x = 262682$:
$p − 1 = 6b$ where $b = 166667$.
Compute $\log_{5^6}(x^6) = 160788$.
Compute $x/5^{160788} = 1000002$.
Compute $\log_{5^b} 1000002 = 3$.
Then $x = 5^{160788+3b} = 5^{660789}$.

Use rho: $\approx \sqrt{a} + \sqrt{b}$ mults.
Better if \(ab\) factors further:
apply Pohlig-Hellman recursively.
Factors of the group order
Assume 5 has order ab.

Given x, a power of 5:

5\(^a\) has order b, and
x\(^a\) is a power of 5\(^b\).
Compute \(\ell = \log_{5^a} x^a\).

5\(^b\) has order a, and
x= 5\(^b\) is a power of 5\(^a\).
Compute \(m = \log_{5^b}(x/5^\ell)\).

Then \(x = 5^\ell + mb\).

This “Pohlig-Hellman method” converts an order-ab DL into an order-a DL, an order-b DL, and a few exponentiations.

e.g. \(p = 1000003, x = 262682: p - 1 = 6b\) where \(b = 166667\).
Compute \(\log_{5^6}(x^6) = 160788\).
Compute \(x/5^{160788} = 1000002\).
Compute \(\log_{5^b}1000002 = 3\).
Then \(x = 5^{160788 + 3b} = 5^{660789}\).

Use rho: \(\approx \sqrt{a} + \sqrt{b}\) mults.
Better if ab factors further: apply Pohlig-Hellman recursively.

All of the techniques so far apply to elliptic curves.

An elliptic curve over \(F_q\) has \(\approx q\) points, so can compute ECDL \(\approx \sqrt{q}\) elliptic-curve adds.
Need quite large \(q\).

If largest prime divisor of number of points is much smaller than \(q\), then Pohlig-Hellman method computes ECDL more quickly.
Need larger \(q\); or change choice of curve.
This “Pohlig-Hellman method” converts an order-\(ab\) DL into an order-\(a\) DL, an order-\(b\) DL, and a few exponentiations.

Given \(x\), a power of 5:
- \(5^a\) has order \(b\), and \(x^a\) is a power of 5.
- \(5^b\) has order \(a\), and \(x=5^b\) is a power of 5.

Compute \(m = \log_{5^b}(x=5^b)\).

Then \(x = 5^a + mb\).

All of the techniques so far apply to elliptic curves.

An elliptic curve over \(F_q\) has \(\approx q + 1\) points, so can compute ECDL using \(q\) elliptic-curve adds.

If largest prime divisor of number of points is much smaller than \(q\), then Pohlig-Hellman computes ECDL more quickly.

Need quite large \(q\); or change choice of curve.

Use rho: \(\approx \sqrt{a} + \sqrt{b}\) mults.
Better if \(ab\) factors further: apply Pohlig-Hellman recursively.
This “Pohlig-Hellman method” converts an order-$ab$ DL into an order-$a$ DL, an order-$b$ DL, and a few exponentiations.

e.g. $p = 1000003$, $x = 262682$:
$p - 1 = 6b$ where $b = 166667$.
Compute $\log_{5^6}(x^6) = 160788$.
$\log_{5^6} x / 5^{160788} = 1000002$.
Compute $\log_{5^6} 1000002 = 3$.
Then $x = 5^{160788 + 3b} = 5^{660789}$.

Use rho: $\approx \sqrt{a} + \sqrt{b}$ mults.
Better if $ab$ factors further:
apply Pohlig-Hellman recursively.

All of the techniques so far apply to elliptic curves.

An elliptic curve over $\mathbb{F}_q$ has $\approx q + 1$ points so can compute ECDL using $\approx \sqrt{q}$ elliptic-curve adds.

Need quite large $q$.

If largest prime divisor of number of points is much smaller than $q$ then Pohlig-Hellman method computes ECDL more quickly.

Need larger $q$; or change choice of curve.
This “Pohlig-Hellman method” converts an order-$ab$ DL into an order-$a$ DL, an order-$b$ DL, and a few exponentiations.

e.g. $p = 1000003$, $x = 262682$: $p - 1 = 6b$ where $b = 166667$.
Compute $\log_{5^6}(x^6) = 160788$.
Compute $x/5^{160788} = 1000002$.
Compute $\log_{5^b} 1000002 = 3$.
Then $x = 5^{160788+3b} = 5^{660789}$.

Use rho: $\approx \sqrt{a} + \sqrt{b}$ mults.
Better if $ab$ factors further:
apply Pohlig-Hellman recursively.

All of the techniques so far apply to elliptic curves.

An elliptic curve over $\mathbf{F}_q$ has $\approx q + 1$ points
so can compute ECDL using $\approx \sqrt{q}$ elliptic-curve adds.
Need quite large $q$.

If largest prime divisor of number of points is much smaller than $q$
then Pohlig-Hellman method computes ECDL more quickly.
Need larger $q$;
or change choice of curve.
The "Pohlig-Hellman method" converts an order-$ab$ DL into an order-$a$ DL, an order-$b$ DL, and a few exponentiations.

For example, let $p = 1000003$, $x = 262682$: 

Then $6b = 166667$. 

Compute $\log_5 6(x^6) = 160788$. 

Then $\log_5 6(x^6) = 1000002$. 

Compute $\log_5 b 1000002 = 3$. 

Then $\log_5 b 1000002 = 5^{160788+3b} = 5^{660789}$.

All of the techniques so far apply to elliptic curves. 

An elliptic curve over $\mathbb{F}_q$ has $\approx q + 1$ points 

so can compute ECDL using $\approx \sqrt{q}$ elliptic-curve adds. 

Need quite large $q$. 

If largest prime divisor of number of points is much smaller than $q$ then Pohlig-Hellman method computes ECDL more quickly. Need larger $q$; or change choice of curve.

Index calculus

Have generated many group elements $5^{an} + b \mod p$. 

Deduced equations for $n$ from random collisions. 

Index calculus obtains discrete-logarithm equations in a different way. 

Example for $p = 1000003$: 

Can completely factor $3^1 = 3$ as $2^6 5^6$, so $-3^{1}/(p - 1) = 1$. 

so $\log_5 (8) - 6 \log_5 2 - 6 \log_5 5$. 

Any of the techniques so far apply to elliptic curves. 

An elliptic curve over $\mathbb{F}_q$ has $\approx q + 1$ points so can compute ECDL using $\approx \sqrt{q}$ elliptic-curve adds. 

Need quite large $q$. 

If largest prime divisor of number of points is much smaller than $q$ then Pohlig-Hellman method computes ECDL more quickly. Need larger $q$; or change choice of curve.
This "Pohlig-Hellman method" converts an order-$ab$ DL into an order-$a$ DL, an order-$b$ DL, and a few exponentiations.

\[ x = 262682: \]
\[ b = 166667. \]
\[ a \cdot b \cdot \text{mults.} \]
\[ b^x = 1000002. \]
\[ 3^x = 2660789. \]

All of the techniques so far apply to elliptic curves.

An elliptic curve over \( F_q \) has \( \approx q + 1 \) points so can compute ECDL using \( \approx \sqrt{q} \) elliptic-curve adds.

Need quite large \( q \).

If largest prime divisor of number of points is much smaller than \( q \) then Pohlig-Hellman method computes ECDL more quickly.

Need larger \( q \); or change choice of curve.

Index calculus:

Have generated many group elements \( 5^a + b \mod p \).

Deduced equations for \( n \) from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for \( p = 1000003 \):

Can completely factor \( -3/(p - 3) \) as \( -3^1 \).

so \( -3^1 \equiv 2^6 5^6 \) (mod \( p \)) so \( \log_5(-1) + \log_5 2 + 6 \log_5 2 + 6 \log_5 5 \)
This “Pohlig-Hellman method” converts an order-$ab$ DL into an order-$a$ DL, an order-$b$ DL, and a few exponentiations.

For example, let $p = 1000003$, $x = 262682$.

Compute $\log_5 6^{160788} = 1000002$.

Compute $\log_5 b^{160788+3} = \log_5 b^{660789}$.

Use rho: $p^a + p^b$ mults.

Better if $ab$ factors further: apply Pohlig-Hellman recursively.

All of the techniques so far apply to elliptic curves.

An elliptic curve over $\mathbb{F}_q$ has $\approx q + 1$ points so can compute ECDL using $\approx \sqrt{q}$ elliptic-curve adds.

Need quite large $q$.

If largest prime divisor of number of points is much smaller than $q$ then Pohlig-Hellman method computes ECDL more quickly.

Need larger $q$; or change choice of curve.

Index calculus

Have generated many group elements $5^{an+b} \mod p$.

Deduced equations for $n$ from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for $p = 1000003$: Can completely factor $-3/(p - 3)$ as $-3^1/2^65^6$ in $\mathbb{Z}/p\mathbb{Z}$ so $-3^1 \equiv 2^65^6 \pmod{p}$.

So $\log_5(-1) + \log_5 3 \equiv 6 \log_5 2 + 6 \log_5 5 \pmod{p + 1}$.
All of the techniques so far apply to elliptic curves.

An elliptic curve over $\mathbb{F}_q$ has $q + 1$ points so can compute ECDL using $\approx \sqrt{q}$ elliptic-curve adds.

Need quite large $q$.

If largest prime divisor of number of points is much smaller than $q$ then Pohlig-Hellman method computes ECDL more quickly.

Need larger $q$; or change choice of curve.

Index calculus

Have generated many group elements $5^{an+b} \mod p$.

Deduced equations for $n$ from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for $p = 1000003$:

Can completely factor $-3/(p - 3)$ as $-3^1/2^6 5^6$ in $\mathbb{Q}$ so $-3^1 \equiv 2^6 5^6 \pmod{p}$.

so $\log_5(-1) + \log_5 3 \equiv 6 \log_5 2 + 6 \log_5 5 \pmod{p - 1}$. 
All of the techniques so far apply to elliptic curves. An elliptic curve over $\mathbb{F}_q$ has $q + 1$ points so can compute ECDL using elliptic-curve adds. Need quite large $q$. If largest prime divisor of number of points is much smaller than $q$ then Pohlig-Hellman method computes ECDL more quickly. Need larger $q$; or change choice of curve.

Index calculus

Have generated many group elements $5^{an+b} \mod p$. Deduced equations for $n$ from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for $p = 1000003$:

Can completely factor $62 = (p + 62)$ as $2^131^1$

so $\log_5 2 + \log_5 31

Try to completely factor $1/(p + 1)$ etc.

Find factorization of $a=p+a$ as product of powers of $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31$ for each of the following a’s:

5100, 4675, 3128, 403, 368, 147, 3, 62, 957, 2912, 3857, 6877.
Index calculus

Have generated many group elements $5^{an+b} \mod p$. Deduced equations for $n$ from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for $p = 1000003$:
Can completely factor $-3/(p - 3)$ as $-3^1/2^65^6$ in $\mathbb{Q}$ so $-3^1 \equiv 2^65^6 \pmod{p}$ so $\log_5(-1) + \log_5 3 \equiv 6\log_5 2 + 6\log_5 5 \pmod{p - 1}$.

Try to completely factor $1/(p + 1), 2/(p + 1)$, Find factorization as product of powers of $2, 3, 5, 7, 11, 13, 17$ for each of the following:

Can completely factor $62$ as $2^13^1/3^15^111^2$ so $\log_5 2 + \log_5 31 + \log_5 3 + \log_5 5 + 2\log_5 11 + \log_5 19 + \log_5 29$.

5100, 4675, 3128, 403, 368, 147, 3, 62, 957, 2912, 3857, 6877.
Index calculus
Have generated many group elements $5^{an+b} \mod p$.
Deduced equations for $n$ from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for $p = 1000003$:
Can completely factor $-3/(p - 3)$ as $-3^1/2^6 5^6$ in $\mathbb{Q}$
so $-3^1 \equiv 2^6 5^6 \pmod{p}$
so $\log_5(-1) + \log_5 3 \equiv 6 \log_5 2 + 6 \log_5 5 \pmod{p - 1}$.

Can completely factor $62/(p + 62)$ as $2^1 3^1 5^1 11^2 19^1 29^1$
so $\log_5 2 + \log_5 31 \equiv \log_5 3 + \log_5 5 + 2 \log_5 11 + \log_5 19 + \log_5 29 \pmod{p - 1}$.

Try to completely factor $1/(p + 1)$, $2/(p + 2)$, etc.
Find factorization of $a/(p + a)$ as product of powers of $-1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29$ for each of the following $a$'s:
$-5100, -4675, -3128, -403, -368, -147, -3, 62, 957, 2912, 3857, 6877$. 

All of the techniques so far apply to elliptic curves.
An elliptic curve over $F_q$ has $q + 1$ points so can compute ECDL using elliptic-curve adds.

Need quite large $q$.
If largest prime divisor of number of points is much smaller than $q$ then Pohlig-Hellman method computes ECDL more quickly.

Need larger $q$; or change choice of curve.
Index calculus

Have generated many group elements $5^{an+b} \mod p$. Deduced equations for $n$ from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for $p = 1000003$:

Can completely factor $62 = (p + 62)$ as $2^1 31^1 / 3^1 5^1 11^2 19^1 29^1$
so $\log_5 2 + \log_5 31 \equiv \log_5 3 + \log_5 5 + 2 \log_5 11 + \log_5 19 + \log_5 29 \pmod{p - 1}$.

Try to completely factor $1/(p + 1), 2/(p + 2)$, etc.

Find factorization of $a/(p + a)$ as product of powers of $-1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31$
for each of the following $a$'s:

$-5100, -4675, -3128, -403, -368, -147, -3, 62, 957, 2912, 3857, 6877$. 

Can completely factor $62/(p + 62)$ as $2^1 31^1 / 3^1 5^1 11^2 19^1 29^1$
so $\log_5 2 + \log_5 31 \equiv \log_5 3 + \log_5 5 + 2 \log_5 11 + \log_5 19 + \log_5 29 \pmod{p - 1}$.
Index calculus

Have generated many group elements $5^{an+b} \mod p$.

Deduced equations for $n$ from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for $p = 1000003$:

Can completely factor $62/(p + 62)$ as $2^131^1/3^15^111^219^129^1$

so $\log_5 2 + \log_5 31 \equiv \log_5 3 + \log_5 5 + 2 \log_5 11 + \log_5 19 + \log_5 29 \pmod{p - 1}$.

Try to completely factor $1/(p + 1), 2/(p + 2)$, etc.

Find factorization of $a/(p + a)$ as product of powers of $-1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31$ for each of the following $a$'s:

$-5100, -4675, -3128, -403, -368, -147, -3, 62, 957, 2912, 3857, 6877$.

Each complete factorization produces a log equation.

Now have 12 linear equations for $\log_5 2, \log_5 3, \ldots$:

Free equations: $\log_5 5 = 1$, $\log_5 (-1) + \log_5 3 \equiv -1 + 6 \log_5 5 \pmod{p - 1}$.

By linear algebra compute $\log_5 2, \log_5 3, \ldots$.

(If this hadn't been enough, could have searched more $a$'s.)

By similar technique obtain discrete log of any target.
Index calculus

Have generated many group elements \( n + b \mod p \).

Deduced equations for \( n \) from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for \( p = 1000003 \):

Can completely factor \( 62/(p + 62) \) as \( 2^13^11^15^111^219^129^1 \)

so \( \log_5 2 + \log_5 31 \equiv \)

\( \log_5 3 + \log_5 5 + 2\log_5 11 + \)

\( \log_5 19 + \log_5 29 \pmod{p - 1}. \)

Try to completely factor \( 1/(p + 1), 2/(p + 2), \) etc.

Find factorization of \( a/(p + a) \) as product of powers of \(-1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31 \)

for each of the following \( a \)'s:

\(-5100, -4675, -3128, -403, -368, -147, -3, 62, 957, 2912, 3857, 6877.\)

Each complete factorization produces a log equation.

Now have 12 linear equations for \( \log_5 2, \log_5 3, \ldots \).

Free equations: \( \log_5(-1) = (p - 1) \).

By linear algebra compute \( \log_5 2, \log_5 3, \ldots, \log_5 31. \)

(If this hadn’t been enough, could have searched more \( a \)'s.)

By similar technique obtain discrete log of any target.
Each complete factorization produces a log equation.

Now have 12 linear equations for log₅ 2, log₅ 3, ..., log₅ 31.

Free equations: log₅ 5 = 1, log₅ (−1) = (p − 1)/2.

By linear algebra compute log₅ 2, log₅ 3, ..., log₅ 31.

(If this hadn’t been enough, could have searched more a’s.)

By similar technique obtain discrete log of any target.

Can completely factor 62/(p + 62) as 2¹3¹1¹/3¹5¹1¹2¹19¹29¹
so log₅ 2 + log₅ 31 ≡ log₅ 3 + log₅ 5 + 2 log₅ 11 + log₅ 19 + log₅ 29 (mod p − 1).

Try to completely factor
1/(p + 1), 2/(p + 2), etc.
Find factorization of a/(p + a) as product of powers of −1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31 for each of the following a’s:
−5100, −4675, −3128, −403, −368, −147, −3, 62, 957, 2912, 3857, 6877.
Can completely factor $62/(p + 62)$ as $2^131^1/3^15^111^219^129^1$
so $\log_5 2 + \log_5 31 \equiv
\log_5 3 + \log_5 5 + 2 \log_5 11 +
\log_5 19 + \log_5 29 \pmod{p - 1}$.

Try to completely factor
$1/(p + 1), 2/(p + 2)$, etc.
Find factorization of $a/(p + a)$
as product of powers of $-1,$ $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31$
for each of the following $a$'s:
$-5100, -4675, -3128,$
$-403, -368, -147, -3,$
$62, 957, 2912, 3857, 6877.$

Each complete factorization
produces a log equation.
Now have 12 linear equations
for $\log_5 2, \log_5 3, \ldots, \log_5 31$.
Free equations: $\log_5 5 = 1,$
$\log_5(-1) = (p - 1)/2$.

By linear algebra compute
$\log_5 2, \log_5 3, \ldots, \log_5 31$.
(If this hadn’t been enough,
could have searched more $a$’s.)

By similar technique obtain
discrete log of any target.
Can completely factor $62/(p + 62)$ as $3^{1}5^{1}11^{1}19^{1}29^{1}$.  

Each complete factorization produces a log equation.

Now have 12 linear equations for $\log_{5} 2, \log_{5} 3, \ldots, \log_{5} 31$.
Free equations: $\log_{5} 2 = 1$, $\log_{5} (-1) = (p - 1)/2$.
By linear algebra compute $\log_{5} 2; \log_{5} 3; \ldots; \log_{5} 31$.

For $p \rightarrow \infty$, index calculus scales surprisingly well: cost $p^{e}$ with $e < 0$.

Each complete factorization produces a log equation.

Now have 12 linear equations for $\log_{5} 2, \log_{5} 3, \ldots, \log_{5} 31$.
Free equations: $\log_{5} 2 = 1$, $\log_{5} (-1) = (p - 1)/2$.
By linear algebra compute $\log_{5} 2, \log_{5} 3, \ldots, \log_{5} 31$.

By similar technique obtain discrete log of any target.

For $p \rightarrow \infty$, index calculus scales surprisingly well: cost $p^{e}$ with $e < 0$.

Specifically: searching $a \in \{-1, 2, 3, \ldots, y\}$ finds $y$ complete factorizations into primes and computes discrete logs.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

(If this hadn’t been enough, could have searched more $a$’s.)
Can completely factor $62 = (p + 62)$ as $2^1 3^1 5^1 11^2 19^1 29^1$.

$1 \equiv 2 \log_5 11 + \ldots \pmod{p - 1}$.

Each complete factorization produces a log equation.

Now have 12 linear equations for $\log_5 2, \log_5 3, \ldots, \log_5 31$.

Free equations: $\log_5 5 = 1$, $\log_5(-1) = (p - 1)/2$.

By linear algebra compute $\log_5 2, \log_5 3, \ldots, \log_5 31$.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

For $p \to \infty$, index calculus scales surprisingly well: cost $p^\epsilon$ where $\epsilon \to 0$.

Compare to rho: $a^2 = 2^1$.

Specifically: searching $a \in \{1, 2, \ldots, y^2\}$ for $\log_5 2, \log_5 3, \ldots, \log_5 31$ finds $y$ complete factorizations into primes $\leq y$, and computes discrete logs.

(Assuming standard conjectures. Have extensive evidence.)

For $p \to 1$, index calculus scales surprisingly well: $\text{cost } p^n$ where $n \to 0$.

Compare to rho: $a^2 = 2^1$.

Specifically: searching $a \in \{1, 2, \ldots, y^2\}$ for $\log_5 2, \log_5 3, \ldots, \log_5 31$ finds $y$ complete factorizations into primes $\leq y$, and computes discrete logs.

(Assuming standard conjectures. Have extensive evidence.)
Each complete factorization produces a log equation.

Now have 12 linear equations for \( \log_5 2, \log_5 3, \ldots, \log_5 31. \)

Free equations: \( \log_5 5 = 1, \)
\( \log_5(-1) = (p - 1)/2. \)

By linear algebra compute \( \log_5 2, \log_5 3, \ldots, \log_5 31. \)

(If this hadn’t been enough, could have searched more \( \alpha \)’s.)

By similar technique obtain discrete log of any target.

For \( p \to \infty, \) index calculus scales surprisingly well: cost \( p^\epsilon \) where \( \epsilon \to 0. \)

Compare to rho: \( \approx p^{1/2}. \)

Specifically: searching \( \alpha \in \{1, 2, \ldots, y^2\}, \) with \( \lg y \in O(\sqrt{\lg p \lg \lg p}), \)
finds \( y \) complete factorizations into primes \( \leq y, \)
and computes discrete logs.

(Assuming standard conjectures. Have extensive evidence.)
Each complete factorization produces a log equation.

Now have 12 linear equations for $\log_5 2, \log_5 3, \ldots, \log_5 31$.
Free equations: $\log_5 5 = 1, \log_5 (-1) = (p - 1)/2$.

By linear algebra compute $\log_5 2, \log_5 3, \ldots, \log_5 31$.

(If this hadn’t been enough, could have searched more $a$’s.)

By similar technique obtain discrete log of any target.

For $p \to \infty$, index calculus scales surprisingly well:
$\text{cost } p^\epsilon$ where $\epsilon \to 0$.

Compare to rho: $\approx p^{1/2}$.

Specifically: searching $a \in \{1, 2, \ldots, y^2\}$, with $\lg y \in O(\sqrt{\lg p \lg \lg p})$, finds $y$ complete factorizations into primes $\leq y$, and computes discrete logs.

(Assuming standard conjectures. Have extensive evidence.)
Each complete factorization produces a log equation.

Now have 12 linear equations for log \(5^2, \log_5 3, \ldots, \log_5 31\).

By linear algebra compute \(\log_5 2, \log_5 3; \ldots\).

If this hadn’t been enough, could have searched more \(a\)’s.

By similar technique obtain discrete log of any target.

For \(p \to \infty\), index calculus scales surprisingly well:
\[
\text{cost } p^\epsilon \text{ where } \epsilon \to 0.
\]

Compare to rho: \(\approx p^{1/2}\).

Specifically: searching \(a \in \{1, 2, \ldots, y^2\}\), with
\[
\lg y \in O\left(\sqrt{\lg p \lg \lg p}\right),
\]
finds \(y\) complete factorizations into primes \(\leq y\),
and computes discrete logs.

(Assuming standard conjectures. Have extensive evidence.)

Latest index-calculus variants use the “number-field sieve”
and the “function-field sieve.”

To compute \(\lg 2^{128}\) the cost is \(\in O\left((\lg q)/3\right)^{1/3}\).

For security:
\(q \approx 2^{256}\) to stop rho; \(q \approx 2^{2048}\) to stop NFS.

We don’t know any index-calculus methods for ECDL!

\[\cdots\] except for some curves.
Each complete factorization produces a log equation.

Now have 12 linear equations for \( \log_5 2 \); \( \log_5 3 \); \ldots; \( \log_5 31 \).

Free equations: \( \log_5 5 = 1 \), \( \log_5 (\frac{1}{2}) = \frac{1}{2} \).

By linear algebra compute \( \log_5 2 \); \( \log_5 3 \); \ldots; \( \log_5 31 \).

(If this hadn’t been enough, could have searched more \( a \)’s.)

By similar technique obtain discrete log of any target.

For \( p \to \infty \), index calculus scales surprisingly well:
\[
\text{cost } p^\epsilon \quad \text{where } \epsilon \to 0.
\]

Compare to rho: \( \approx p^{1/2} \).

Specifically: searching \( a \in \{1, 2, \ldots, y^2\} \), with \( \lg y \in O(\sqrt{\lg p \lg \lg p}) \), finds \( y \) complete factorizations into primes \( \leq y \), and computes discrete logs.

(Assuming standard conjectures. Have extensive evidence.)

Latest index-calculus variants use the “number-field sieve” and the “function-field sieve.”

To compute discrete logs in \( \text{F}_q \):
\[
\lg \text{cost} \in O((\lg q)^{1/3}(\lg \lg q)).
\]

For security:
\( q \approx 2^{256} \) to stop rho;
\( q \approx 2^{2048} \) to stop NFS.

We don’t know any index-calculus methods for ECDL! … except for some.
For $p \to \infty$, index calculus scales surprisingly well: 
\[ \text{cost } p^\epsilon \text{ where } \epsilon \to 0. \]

Compare to rho: \( \approx p^{1/2} \).

Specifically: searching
\[ a \in \{1, 2, \ldots, y^2\}, \]
with
\[ \lg y \in O(\sqrt{\lg p \lg \lg p}), \]
finds $y$ complete factorizations into primes \( \leq y \),
and computes discrete logs.

( Assuming standard conjectures. Have extensive evidence.)

Latest index-calculus variants use the “number-field sieve” and the “function-field sieve.”

To compute discrete logs in \( \mathbb{F}_q \):
\[ \lg \text{cost } \in O((\lg q)^{1/3}(\lg \lg q)^{2/3}). \]

For security:
\[ q \approx 2^{256} \text{ to stop rho; } \]
\[ q \approx 2^{2048} \text{ to stop NFS.} \]

We don’t know any index-calculus methods for ECDL!
\[ \text{... except for some curves.} \]
For $p \rightarrow \infty$, index calculus scales surprisingly well: 
\[ \text{cost } p^\epsilon \text{ where } \epsilon \rightarrow 0. \]

Compare to rho: $\approx p^{1/2}$.

Specifically: searching $a \in \{1, 2, \ldots, y^2\}$, with 
\[ \lg y \in O(\sqrt{\lg p \lg \lg p}), \]
finds $y$ complete factorizations into primes $\leq y$, 
and computes discrete logs.

(Assuming standard conjectures. Have extensive evidence.)

Latest index-calculus variants use the “number-field sieve” 
and the “function-field sieve.”

To compute discrete logs in $\mathbb{F}_q$:
\[ \lg \text{cost} \in O((\lg q)^{1/3} (\lg \lg q)^{2/3}). \]

For security:
$q \approx 2^{256}$ to stop rho;
$q \approx 2^{2048}$ to stop NFS.

We don’t know any 
index-calculus methods for ECDL!
\[ \ldots \text{ except for some curves.} \]