Towards an exact cost analysis of index-calculus algorithms

ECC 2006

Nicolas Thériault
ntheriau@fields.utoronto.ca
Fields Institute

Work in progress with *Roberto Avanzi*

For an elliptic curve $E(\mathbb{F}_{\tilde{q}})$, the fastest known method to solve the discrete logarithm problem is Pollard's Rho algorithm, which takes an expected

$$\left(\frac{\sqrt{\pi}}{2} + \epsilon\right)\sqrt{\text{group order}} \quad \text{group operations.}$$

For an elliptic curve $E(\mathbb{F}_{\tilde{q}})$, the fastest known method to solve the discrete logarithm problem is Pollard's Rho algorithm, which takes an expected

$$\left(\frac{\sqrt{\pi}}{2}+\epsilon\right)\sqrt{\text{group order}} \quad \text{group operations.}$$

If ℓ is the largest prime divisor of the group order, this means $\approx 0.8862\sqrt{\ell}$ group operations (Pohlig–Hellman).

For an elliptic curve $E(\mathbb{F}_{\tilde{q}})$, the fastest known method to solve the discrete logarithm problem is Pollard's Rho algorithm, which takes an expected

$$\left(\frac{\sqrt{\pi}}{2} + \epsilon\right) \sqrt{\text{group order}} \quad \text{group operations.}$$

If ℓ is the largest prime divisor of the group order, this means $\approx 0.8862\sqrt{\ell}$ group operations (Pohlig–Hellman).

This is often stated as $O(\sqrt{\tilde{q}})$, hiding the constant $\frac{\sqrt{\pi}}{2}$ (assuming a group of prime order).

Genus 2 HEC DLP

For a hyperelliptic curves H of genus 2 defined over the field $\mathbb{F}_q$, the fastest known method to solve the discrete log is still Pollard Rho, which is often stated as $O(q)$.

For a hyperelliptic curves H of genus 2 defined over the field $\mathbb{F}_q$, the fastest known method to solve the discrete log is still Pollard Rho, which is often stated as $O(q)$.

To get the same security as an elliptic curve over $\mathbb{F}_{\tilde{q}}$, we ask that q has half the number of bits as $\tilde{q}$ (abusing notation, we ask that $O(\sqrt{\tilde{q}}) = O(q)$).

We then compare the efficiency of $Jac(H)(\mathbb{F}_q)$ with $E(\mathbb{F}_{\tilde{q}})$.

Genus 2 HEC DLP

For a hyperelliptic curves H of genus 2 defined over the field $\mathbb{F}_q$, the fastest known method to solve the discrete log is still Pollard Rho, which is often stated as $O(q)$.

To get the same security as an elliptic curve over $\mathbb{F}_{\tilde{q}}$, we ask that q has half the number of bits as $\tilde{q}$ (abusing notation, we ask that $O(\sqrt{\tilde{q}}) = O(q)$).

We then compare the efficiency of $Jac(H)(\mathbb{F}_q)$ with $E(\mathbb{F}_{\tilde{q}})$.

Since the security of both group depends on the same algorithm, the hidden constants are the same, so we might expect a fair comparison.

But we are making a very big assumption: the group operations have the same cost when we attack the DLP as they have in the scalar multiplication.

Group Operations

But we are making a very big assumption: the group operations have the same cost when we attack the DLP as they have in the scalar multiplication.

This might not be true!

The fastest algorithm to do the group operation for the scalar multiplication might not the the best choice for Pollard Rho (Montgomery's trick for inversion, etc).

But we are making a very big assumption: the group operations have the same cost when we attack the DLP as they have in the scalar multiplication.

This might not be true!

The fastest algorithm to do the group operation for the scalar multiplication might not the the best choice for Pollard Rho (Montgomery's trick for inversion, etc).

We need to take into account the difference in the cost of the group operations.

Convention: We define the average group operation as the average cost per group operation in a NAF.

Genus 1 vs Genus 2

In characteristic 2, we find

$$\approx 0.5653\sqrt{\tilde{q}}$$

for $E(\mathbb{F}_{\tilde{q}})$ of order $2 \cdot \text{prime}$ at 157 bits, and at 223 bits:

$$\approx 0.5625\sqrt{\tilde{q}} \quad .$$

Genus 1 vs Genus 2

In characteristic 2, we find

$$\approx 0.5653\sqrt{\tilde{q}}$$

for $E(\mathbb{F}_{\tilde{q}})$ of order $2 \cdot$ prime at 157 bits, and at 223 bits:

$$\approx 0.5625\sqrt{\tilde{q}} \quad .$$

For genus 2, we find

$$\approx 0.5283\sqrt{q}$$

for $Jac(H)(\mathbb{F}_q)$ of order $2 \cdot$ prime at 79 bits and at 109 bits:

$$\approx 0.5348\sqrt{q} \quad .$$

Genus g

When we look at hyperelliptic curves H of genus $g > 2$ (but not too large) over $\mathbb{F}_q$, asymptotically, Pollard Rho is not the fastest method to solve the discrete log, index calculus is.

Genus g

When we look at hyperelliptic curves H of genus $g > 2$ (but not too large) over $\mathbb{F}_q$, asymptotically, Pollard Rho is not the fastest method to solve the discrete log, index calculus is.

Different versions of index calculus have different running times, for example:

$$O\left(q^{2-\frac{2}{g}+\epsilon}\right)$$

for the double large prime method (2LP)

When we look at hyperelliptic curves H of genus $g > 2$ (but not too large) over $\mathbb{F}_q$, asymptotically, Pollard Rho is not the fastest method to solve the discrete log, index calculus is.

Different versions of index calculus have different running times, for example:

$$O\left(q^{2-\frac{2}{g}+\epsilon}\right)$$

for the double large prime method (2LP), and

$$O\left(q^{2-\frac{2}{g+1/2}+\epsilon}\right)$$

for the single large prime method (1LP).

Problems

It is very difficult to obtain a fair comparison with ECC if we only know the asymptotic form.

We could be dealing with constants 1000, 7.23, 0.002, etc, and these constants could change from one method to the other (and in fact they do).

Problems

It is very difficult to obtain a fair comparison with ECC if we only know the asymptotic form.

We could be dealing with constants 1000, 7.23, 0.002, etc, and these constants could change from one method to the other (and in fact they do).

The impact of index calculus at cryptographic sizes depends a lot on the value of the constant.

The smaller the constants are, the lower the security of $Jac(H)(\mathbb{F}_q)$ really is. If the constant is too large Pollard Rho could still be the fastest method to solve the DLP.

Problems

It is very difficult to obtain a fair comparison with ECC if we only know the asymptotic form.

We could be dealing with constants 1000, 7.23, 0.002, etc, and these constants could change from one method to the other (and in fact they do).

The impact of index calculus at cryptographic sizes depends a lot on the value of the constant.

The smaller the constants are, the lower the security of $Jac(H)(\mathbb{F}_q)$ really is. If the constant is too large Pollard Rho could still be the fastest method to solve the DLP.

Index calculus does not exclude security, but it could exclude efficiency!

Problems

Index calculus *should* be nicer to analyze: the standard deviation of the running time is tiny when compared with the expected value (by an order of magnitude).

In comparison, for Pollard Rho we have a standard deviation of almost $1/2$ of the expected value.

Problems

Index calculus *should* be nicer to analyze: the standard deviation of the running time is tiny when compared with the expected value (by an order of magnitude).

In comparison, for Pollard Rho we have a standard deviation of almost $1/2$ of the expected value.

But there is currently no detailed analysis of the constant for index calculus available in literature:

Problems

Index calculus *should* be nicer to analyze: the standard deviation of the running time is tiny when compared with the expected value (by an order of magnitude).

In comparison, for Pollard Rho we have a standard deviation of almost $1/2$ of the expected value.

But there is currently no detailed analysis of the constant for index calculus available in literature:

- The algorithm is more complicated
- Not all costs involved are group operations

Problems

Index calculus *should* be nicer to analyze: the standard deviation of the running time is tiny when compared with the expected value (by an order of magnitude).

In comparison, for Pollard Rho we have a standard deviation of almost $1/2$ of the expected value.

But there is currently no detailed analysis of the constant for index calculus available in literature:

- The algorithm is more complicated
- Not all costs involved are group operations
- We haven't finished writing it up...

Asymptotically, the 2LP method is faster than the 1LP method by a factor of $O(q^{1/(g^2+g/2)+\epsilon})$.

1LP vs 2LP

Asymptotically, the 2LP method is faster than the 1LP method by a factor of $O(q^{1/(g^2+g/2)+\epsilon})$.

But:

- Even rough estimates give a bigger constant for 2LP.
- The analysis is tighter for 1LP.
- For genus ≥ 4, 1LP could win.

1LP vs 2LP

Asymptotically, the 2LP method is faster than the 1LP method by a factor of $O(q^{1/(g^2+g/2)+\epsilon})$.

But:

- Even rough estimates give a bigger constant for 2LP.
- The analysis is tighter for 1LP.
- For genus ≥ 4, 1LP could win.
- At cryptographic sizes, some of the approximations used to obtain the asymptotic form have a big impact on the constant.
- One improvement does not apply to 2LP (so far).

To simplify notation on the slides, we let $g = 4$.

To simplify notation on the slides, we let $g = 4$.

The cost of the single large prime index calculus is dominated by two things:

- The random Walk (to find 1-almost-smooth divisors)
- The Linear algebra solver (find a vector of the kernel)

Analysis of 1LP

To simplify notation on the slides, we let $g = 4$.

The cost of the single large prime index calculus is dominated by two things:

- The random Walk (to find 1-almost-smooth divisors)
- The Linear algebra solver (find a vector of the kernel)

which gives, for a factor base of size B:

$$T(B) = (1+\epsilon)\left(c_W \cdot 3\sqrt{2}B^{-5/2}q^{7/2} + c_L \cdot \frac{3}{2}B^2\right)$$

Analysis of 1LP

To simplify notation on the slides, we let $g = 4$.

The cost of the single large prime index calculus is dominated by two things:

- The random Walk (to find 1-almost-smooth divisors)
- The Linear algebra solver (find a vector of the kernel)

which gives, for a factor base of size B:

$$T(B) = (1 + \epsilon)\left(c_W \cdot 3\sqrt{2}B^{-5/2}q^{7/2} + c_L \cdot \frac{3}{2}B^2\right)$$

All other parts of the algorithm can be put in the $+\epsilon$ (so we ignore them).

To minimize, we differentiate $T(B)$, solve $\frac{d}{dB}T(B) = 0$ and substitute back into $T(B)$.

Analysis of 1LP

To minimize, we differentiate $T(B)$, solve $\frac{d}{dB}T(B) = 0$ and substitute back into $T(B)$.

After playing around, we obtain a minimum of

$$T_{min} = 3\left(\frac{2^{7/9}}{5^{5/9}} + \frac{5^{4/9}}{2^{11/9}}\right) {c_L}^{5/9} {c_W}^{4/9} q^{14/9}$$

To minimize, we differentiate $T(B)$, solve $\frac{d}{dB}T(B) = 0$ and substitute back into $T(B)$.

After playing around, we obtain a minimum of

$$T_{min} = 3\left(\frac{2^{7/9}}{5^{5/9}} + \frac{5^{4/9}}{2^{11/9}}\right) {c_L}^{5/9} {c_W}^{4/9} q^{14/9}$$

To find the exact cost of 1LP, all we need is c_W and c_L...

To minimize, we differentiate $T(B)$, solve $\frac{d}{dB}T(B) = 0$ and substitute back into $T(B)$.

After playing around, we obtain a minimum of

$$T_{min} = 3\left(\frac{2^{7/9}}{5^{5/9}} + \frac{5^{4/9}}{2^{11/9}}\right) {c_L}^{5/9} {c_W}^{4/9} q^{14/9}$$

To find the exact cost of 1LP, all we need is c_W and c_L...

To improve the running time, we can:

To minimize, we differentiate $T(B)$, solve $\frac{d}{dB}T(B) = 0$ and substitute back into $T(B)$.

After playing around, we obtain a minimum of

$$T_{min} = 3\left(\frac{2^{7/9}}{5^{5/9}} + \frac{5^{4/9}}{2^{11/9}}\right) {c_L}^{5/9} {c_W}^{4/9} q^{14/9}$$

To find the exact cost of 1LP, all we need is c_W and c_L...

To improve the running time, we can:

1. Improve c_L

Analysis of 1LP

To minimize, we differentiate $T(B)$, solve $\frac{d}{dB}T(B) = 0$ and substitute back into $T(B)$.

After playing around, we obtain a minimum of

$$T_{min} = 3\left(\frac{2^{7/9}}{5^{5/9}} + \frac{5^{4/9}}{2^{11/9}}\right) {c_L}^{5/9} {c_W}^{4/9} q^{14/9}$$

To find the exact cost of 1LP, all we need is c_W and c_L...

To improve the running time, we can:

1. Improve c_L
2. Improve c_W

Analysis of 1LP

To minimize, we differentiate $T(B)$, solve $\frac{d}{dB}T(B) = 0$ and substitute back into $T(B)$.

After playing around, we obtain a minimum of

$$T_{min} = 3\left(\frac{2^{7/9}}{5^{5/9}} + \frac{5^{4/9}}{2^{11/9}}\right) {c_L}^{5/9} {c_W}^{4/9} q^{14/9}$$

To find the exact cost of 1LP, all we need is c_W and c_L...

To improve the running time, we can:

1. Improve c_L
2. Improve c_W
3. Introduce a new factor

- Almost-all smooth relations involve 6 points.
- The density of the matrix is very low, Wiedemann is faster than Lanczos.

- Almost-all smooth relations involve 6 points.
- The density of the matrix is very low, Wiedemann is faster than Lanczos.
- All entries are ± 1. If some ± 2 appear in the system (around $O(9)$ times), they are done as two ± 1.
- We use Horner's trick to evaluate $p(M)\mathbf{x}$. (no multiplications!)

Linear Algebra

- Almost-all smooth relations involve 6 points.
- The density of the matrix is very low, Wiedemann is faster than Lanczos.
- All entries are ± 1. If some ± 2 appear in the system (around $O(9)$ times), they are done as two ± 1.
- We use Horner's trick to evaluate $p(M)\mathbf{x}$. (no multiplications!)
- c_L is the cost of all the operations associated to one matrix entry for three matrix–vector multiplications.
- Berlekamp–Massey can be done in sub-quadratic time, so we ignore that cost (for now).

- Almost-all smooth relations involve 6 points.
- The density of the matrix is very low, Wiedemann is faster than Lanczos.
- All entries are ± 1. If some ± 2 appear in the system (around $O(9)$ times), they are done as two ± 1.
- We use Horner's trick to evaluate $p(M)\mathbf{x}$. (no multiplications!)
- c_L is the cost of all the operations associated to one matrix entry for three matrix–vector multiplications.
- Berlekamp–Massey can be done in sub-quadratic time, so we ignore that cost (for now).
- To reduce c_L, we need block Wiedemann (in progress).

Filtering in the Number Field Sieve:
Remove equations and variables that do not contribute to the kernel

- Columns of zeros (variables not in use).
- Singletons (variables appearing only in one equation).
- Removing superfluous equations.

Filtering in the Number Field Sieve:
Remove equations and variables that do not contribute to the kernel

- Columns of zeros (variables not in use).
- Singletons (variables appearing only in one equation).
- Removing superfluous equations.

Merging (preprocessing): if a variable appears in very few equations (2, 3, maybe 4) we can use one of the equations to cancel the variable from the system.

Filtering

Filtering in the Number Field Sieve:
Remove equations and variables that do not contribute to the kernel

- Columns of zeros (variables not in use).
- Singletons (variables appearing only in one equation).
- Removing superfluous equations.

Merging (preprocessing): if a variable appears in very few equations (2, 3, maybe 4) we can use one of the equations to cancel the variable from the system.

- Very limited impact for HEC index calculus.
- In progress...

Extended Filtering

The impact of filtering is quite limited for our systems.
We decided to extend the idea:

The impact of filtering is quite limited for our systems.
We decided to extend the idea:

- We find many more equations (smooth relations) than the number of variables (elements of the factor base)
- For a subset of the variables, we might be able to remove all the equations containing those variables.
- The filtered system can be much smaller than the original system.

Extended Filtering

The impact of filtering is quite limited for our systems.
We decided to extend the idea:

- We find many more equations (smooth relations) than the number of variables (elements of the factor base)
- For a subset of the variables, we might be able to remove all the equations containing those variables.
- The filtered system can be much smaller than the original system.
- Finding the smallest consistent set of variables and equations is most likely NP-hard.
- A good approximation can be found in linear time.

Extended Filtering

The impact of filtering is quite limited for our systems.
We decided to extend the idea:

- We find many more equations (smooth relations) than the number of variables (elements of the factor base)
- For a subset of the variables, we might be able to remove all the equations containing those variables.
- The filtered system can be much smaller than the original system.
- Finding the smallest consistent set of variables and equations is most likely NP-hard.
- A good approximation can be found in linear time.

The result: Harvesting!

Assume that we find k times as many equations as variables, and let $p(k)$ be the proportion of the factor base left after harvesting.

Assume that we find k times as many equations as variables, and let $p(k)$ be the proportion of the factor base left after harvesting.
If k is not too big, we get

$$T_k(B) = c_W \cdot 3\sqrt{2}\sqrt{k}B^{-5/2}q^{7/2} + c_L \cdot \frac{3}{2}p(k)^2B^2$$

and the minimal value is

$$T_{k,min} = 3\left(\frac{2^{7/9}}{5^{5/9}} + \frac{5^{4/9}}{2^{11/9}}\right){c_L}^{5/9}{c_W}^{4/9}\left(kp(k)^5\right)^{2/9}q^{14/9}$$

Assume that we find k times as many equations as variables, and let $p(k)$ be the proportion of the factor base left after harvesting.
If k is not too big, we get

$$T_k(B) = c_W \cdot 3\sqrt{2}\sqrt{k}B^{-5/2}q^{7/2} + c_L \cdot \frac{3}{2}p(k)^2 B^2$$

and the minimal value is

$$T_{k,min} = 3\left(\frac{2^{7/9}}{5^{5/9}} + \frac{5^{4/9}}{2^{11/9}}\right) {c_L}^{5/9} {c_W}^{4/9} \left(kp(k)^5\right)^{2/9} q^{14/9}$$

The good news: $f(k) = kp(k)^5$ is strictly decreasing.

Assume that we find k times as many equations as variables, and let $p(k)$ be the proportion of the factor base left after harvesting.
If k is not too big, we get

$$T_k(B) = c_W \cdot 3\sqrt{2}\sqrt{k}B^{-5/2}q^{7/2} + c_L \cdot \frac{3}{2}p(k)^2B^2$$

and the minimal value is

$$T_{k,min} = 3\left(\frac{2^{7/9}}{5^{5/9}} + \frac{5^{4/9}}{2^{11/9}}\right) {c_L}^{5/9} {c_W}^{4/9} \left(kp(k)^5\right)^{2/9} q^{14/9}$$

The good news: $f(k) = kp(k)^5$ is strictly decreasing.
The bad news: hard to analyze and decreases slowly.

To simplify, we will only use $k = 4$.

Harvesting

To simplify, we will only use $k = 4$.

For $q = 2^{53}$, we have some interference due to the field size, and we need to increase k by 3.29% in the random walk side, giving us

$$\tilde{f}(4)^{2/9} = (4 \cdot 1.0329 \cdot p(4)^5)^{2/9} \approx 0.7142$$

For $q = 2^{73}$, the interference is close to 0, and we are very close to

$$f(4)^{2/9} \approx 0.7090$$

Harvesting

To simplify, we will only use $k = 4$.

For $q = 2^{53}$, we have some interference due to the field size, and we need to increase k by 3.29% in the random walk side, giving us

$$\tilde{f}(4)^{2/9} = (4 \cdot 1.0329 \cdot p(4)^5)^{2/9} \approx 0.7142$$

For $q = 2^{73}$, the interference is close to 0, and we are very close to

$$f(4)^{2/9} \approx 0.7090$$

We get a time-memory trade-off.
Harvesting has a similar impact on 0LP, but we don't know how to predict the impact on 2LP.

For index calculus, the goal of the random walk is very different from its goal in Pollard Rho:

1. Index calculus only cares about the factorization of group elements, not how we get to them (or where we would go afterwards).

For index calculus, the goal of the random walk is very different from its goal in Pollard Rho:

1. Index calculus only cares about the factorization of group elements, not how we get to them (or where we would go afterwards).
2. At each divisor considered, we have to see if it splits completely over $\mathbb{F}_q$.
3. If a divisor splits $(1 : 24)$, we have to factor it and see if it is smooth, 1-almost-smooth, etc.

Random Walk

For index calculus, the goal of the random walk is very different from its goal in Pollard Rho:

1. Index calculus only cares about the factorization of group elements, not how we get to them (or where we would go afterwards).
2. At each divisor considered, we have to see if it splits completely over $\mathbb{F}_q$.
3. If a divisor splits $(1:24)$, we have to factor it and see if it is smooth, 1-almost-smooth, etc.

Point 1 will affect how we do the random walk.

Random Walk

For index calculus, the goal of the random walk is very different from its goal in Pollard Rho:

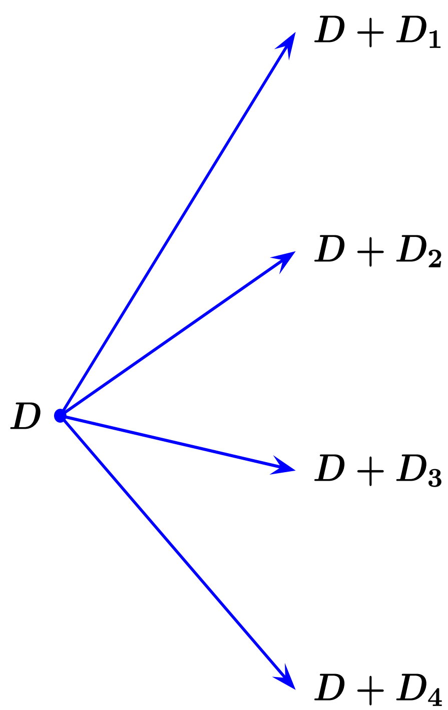
1. Index calculus only cares about the factorization of group elements, not how we get to them (or where we would go afterwards).
2. At each divisor considered, we have to see if it splits completely over $\mathbb{F}_q$.
3. If a divisor splits $(1 : 24)$, we have to factor it and see if it is smooth, 1-almost-smooth, etc.

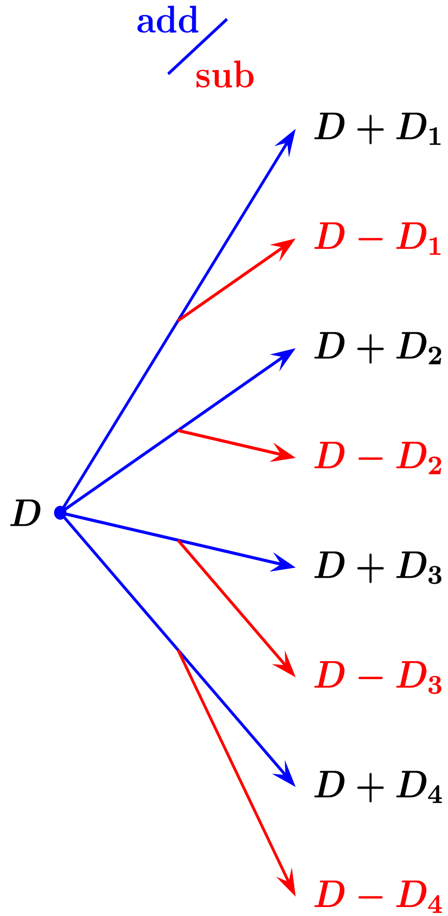
Point 1 will affect how we do the random walk.

Point 2 and 3 have a huge impact on c_W, and they depend on algorithms to factor polynomials.

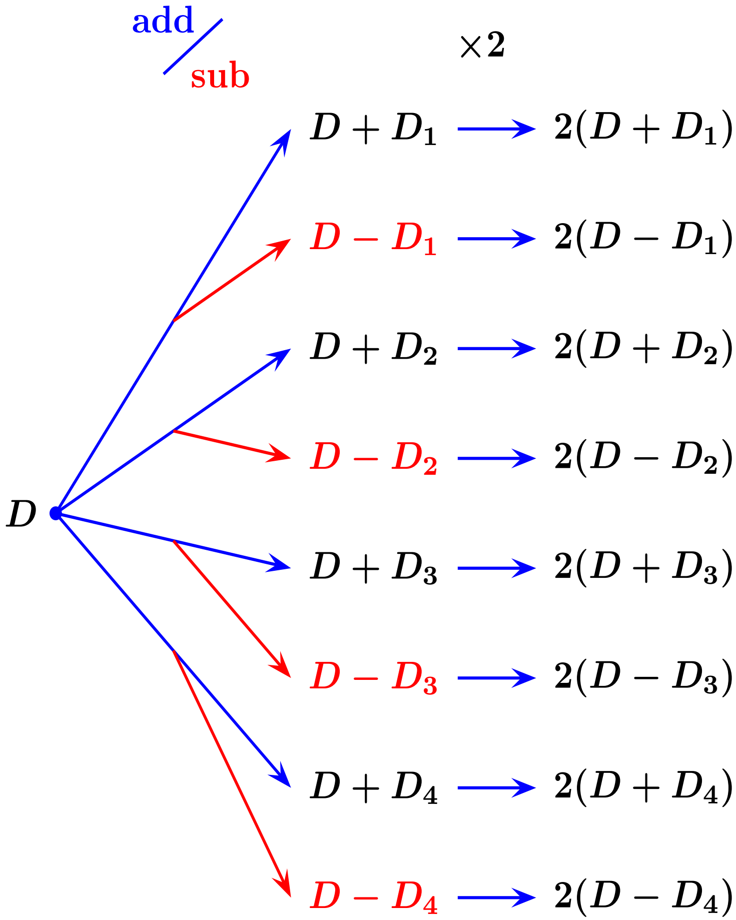
D

add

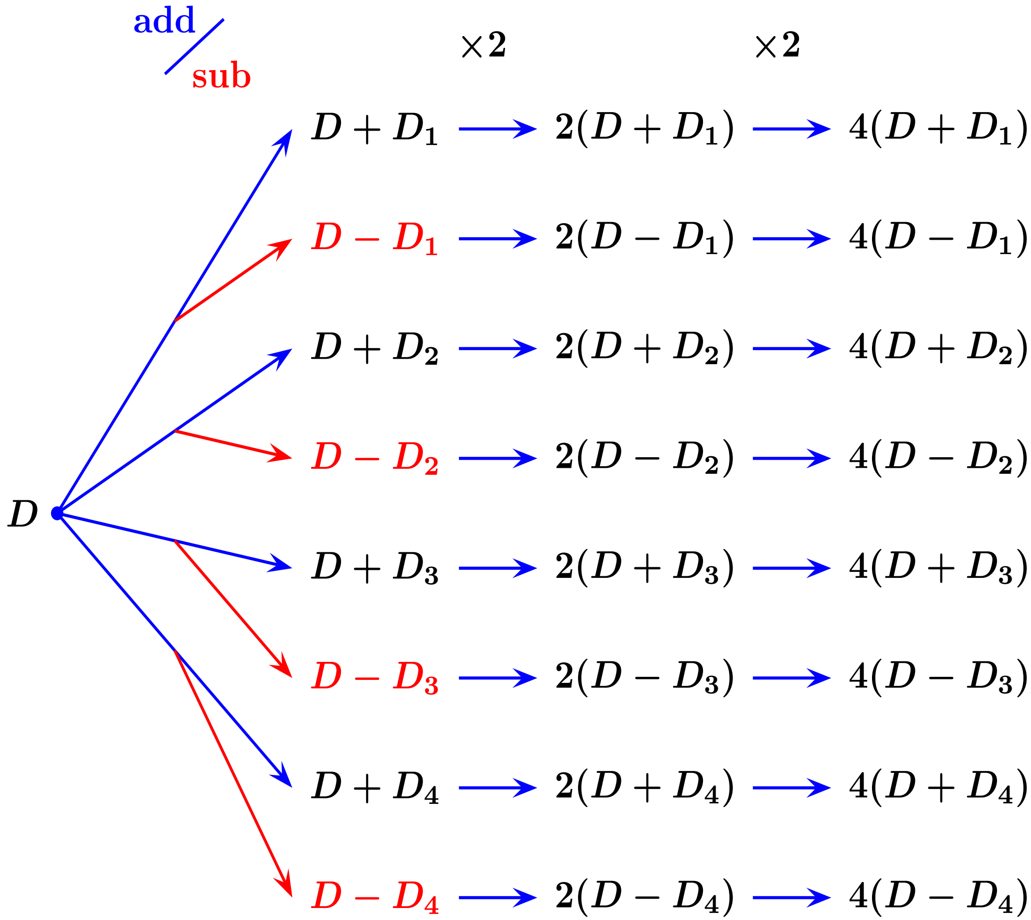



add
sub
$D + D_1$
$D - D_1$
$D + D_2$
$D - D_2$
D
$D + D_3$
$D - D_3$
$D + D_4$
$D - D_4$

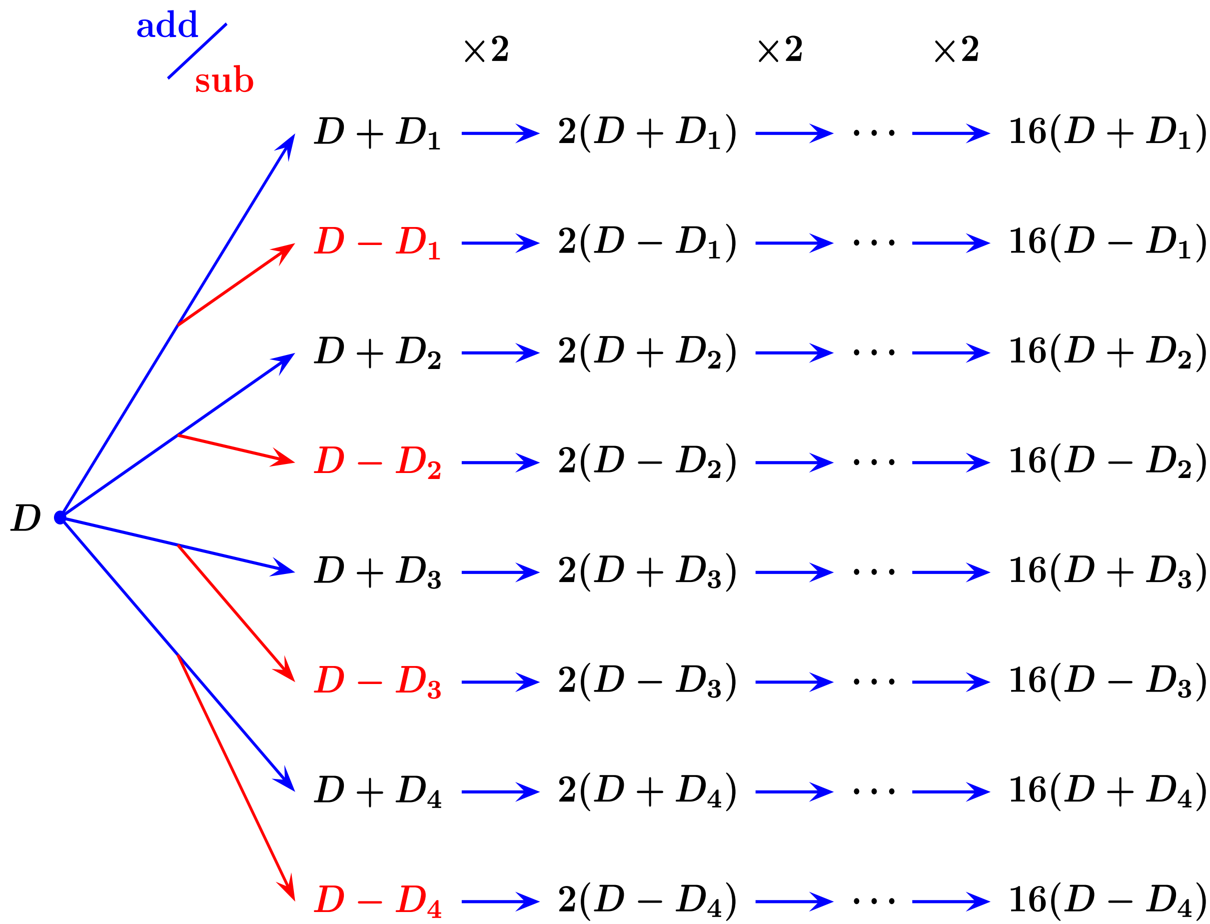
Random Walk

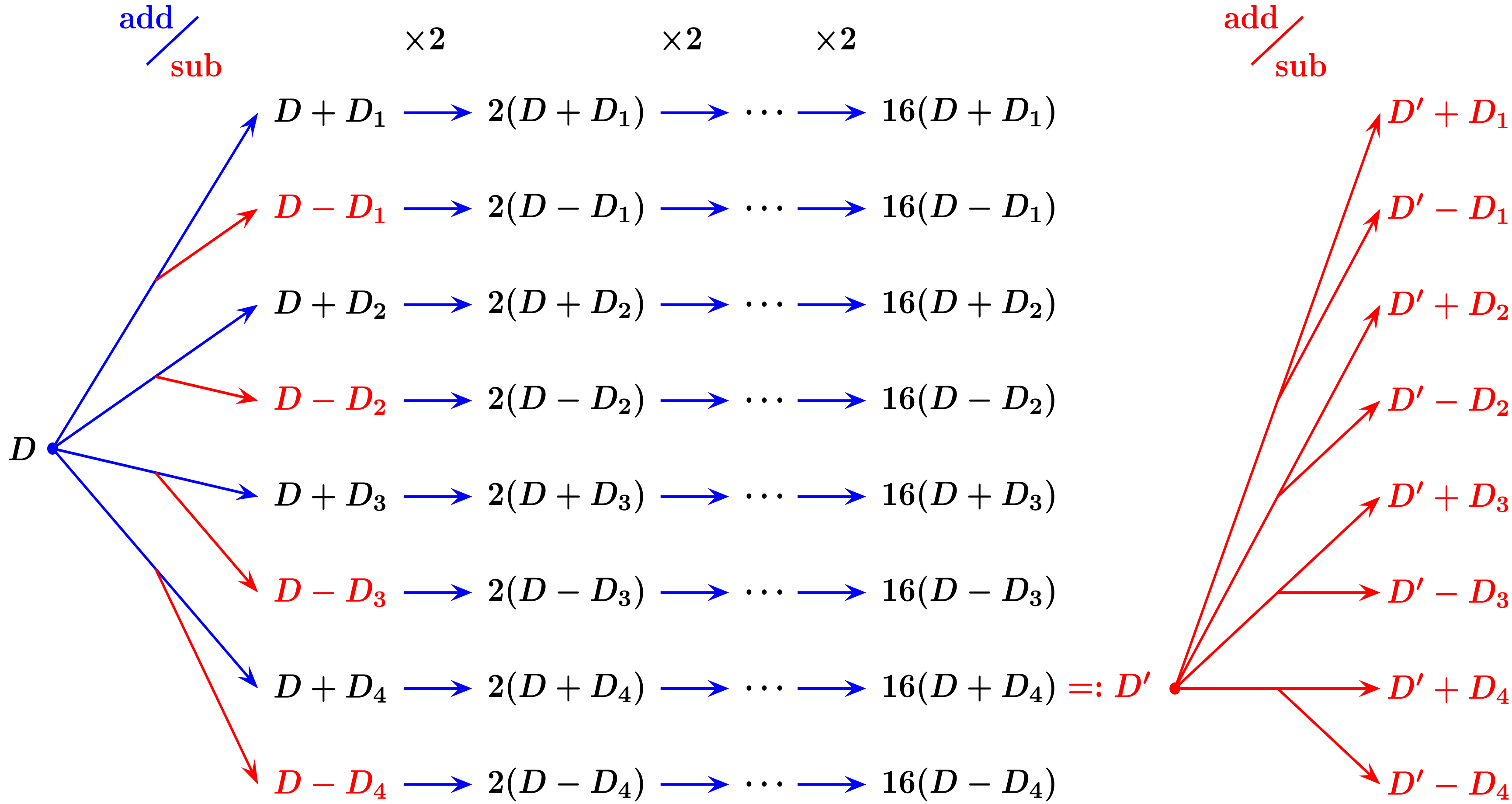

add
sub
×2
D
D + D1
D − D1
D + D2
D − D2
D + D3
D − D3
D + D4
D − D4
2(D + D1)
2(D − D1)
2(D + D2)
2(D − D2)
2(D + D3)
2(D − D3)
2(D + D4)
2(D − D4)

Random Walk

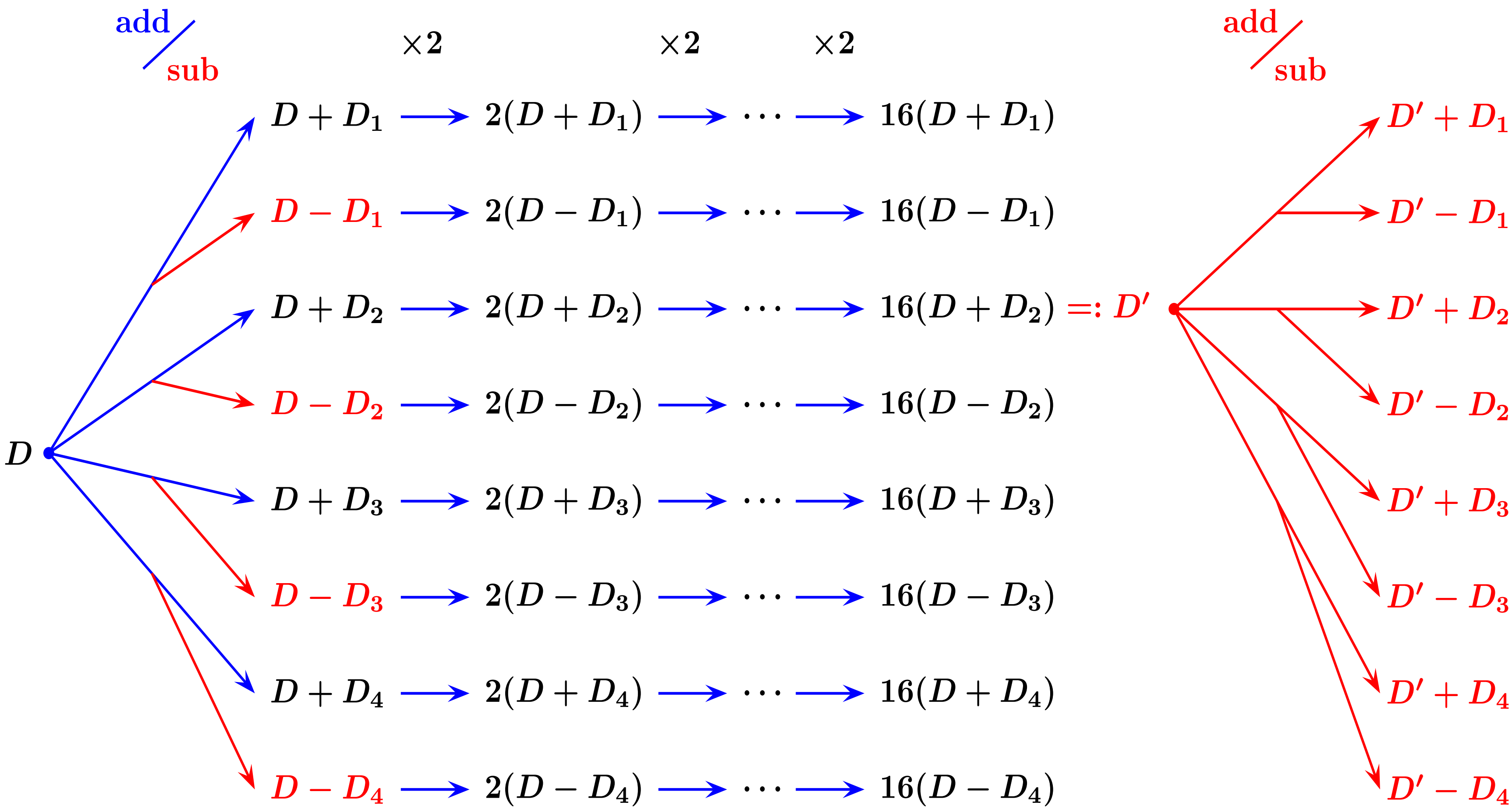

add
sub
×2
×2
D
$D + D_1 \longrightarrow 2(D + D_1) \longrightarrow 4(D + D_1)$
$D - D_1 \longrightarrow 2(D - D_1) \longrightarrow 4(D - D_1)$
$D + D_2 \longrightarrow 2(D + D_2) \longrightarrow 4(D + D_2)$
$D - D_2 \longrightarrow 2(D - D_2) \longrightarrow 4(D - D_2)$
$D + D_3 \longrightarrow 2(D + D_3) \longrightarrow 4(D + D_3)$
$D - D_3 \longrightarrow 2(D - D_3) \longrightarrow 4(D - D_3)$
$D + D_4 \longrightarrow 2(D + D_4) \longrightarrow 4(D + D_4)$
$D - D_4 \longrightarrow 2(D - D_4) \longrightarrow 4(D - D_4)$

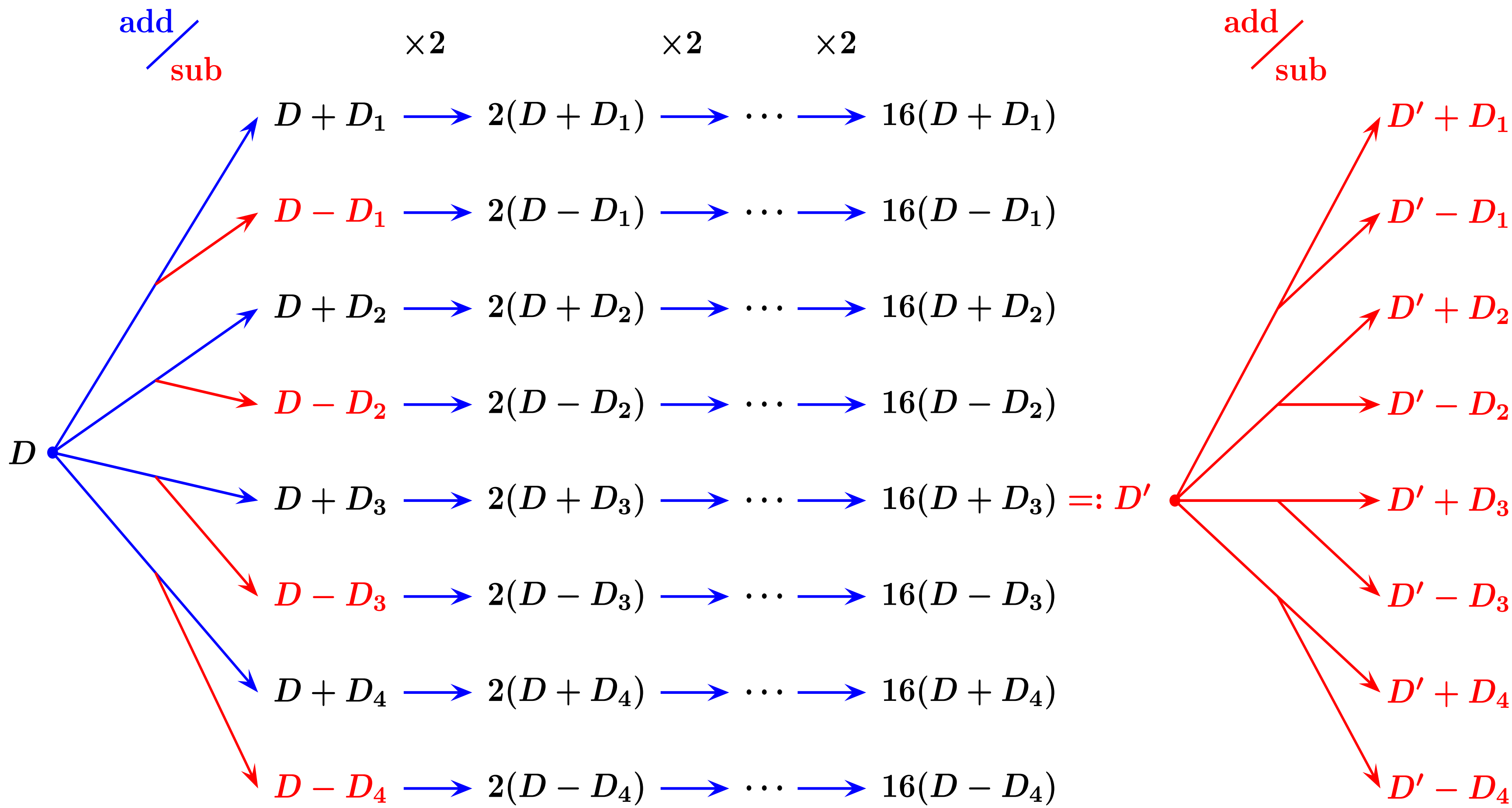
Random Walk


add
sub
×2
×2
×2
D
$D + D_1 \longrightarrow 2(D + D_1) \longrightarrow \cdots \longrightarrow 16(D + D_1)$
$D - D_1 \longrightarrow 2(D - D_1) \longrightarrow \cdots \longrightarrow 16(D - D_1)$
$D + D_2 \longrightarrow 2(D + D_2) \longrightarrow \cdots \longrightarrow 16(D + D_2)$
$D - D_2 \longrightarrow 2(D - D_2) \longrightarrow \cdots \longrightarrow 16(D - D_2)$
$D + D_3 \longrightarrow 2(D + D_3) \longrightarrow \cdots \longrightarrow 16(D + D_3)$
$D - D_3 \longrightarrow 2(D - D_3) \longrightarrow \cdots \longrightarrow 16(D - D_3)$
$D + D_4 \longrightarrow 2(D + D_4) \longrightarrow \cdots \longrightarrow 16(D + D_4)$
$D - D_4 \longrightarrow 2(D - D_4) \longrightarrow \cdots \longrightarrow 16(D - D_4)$

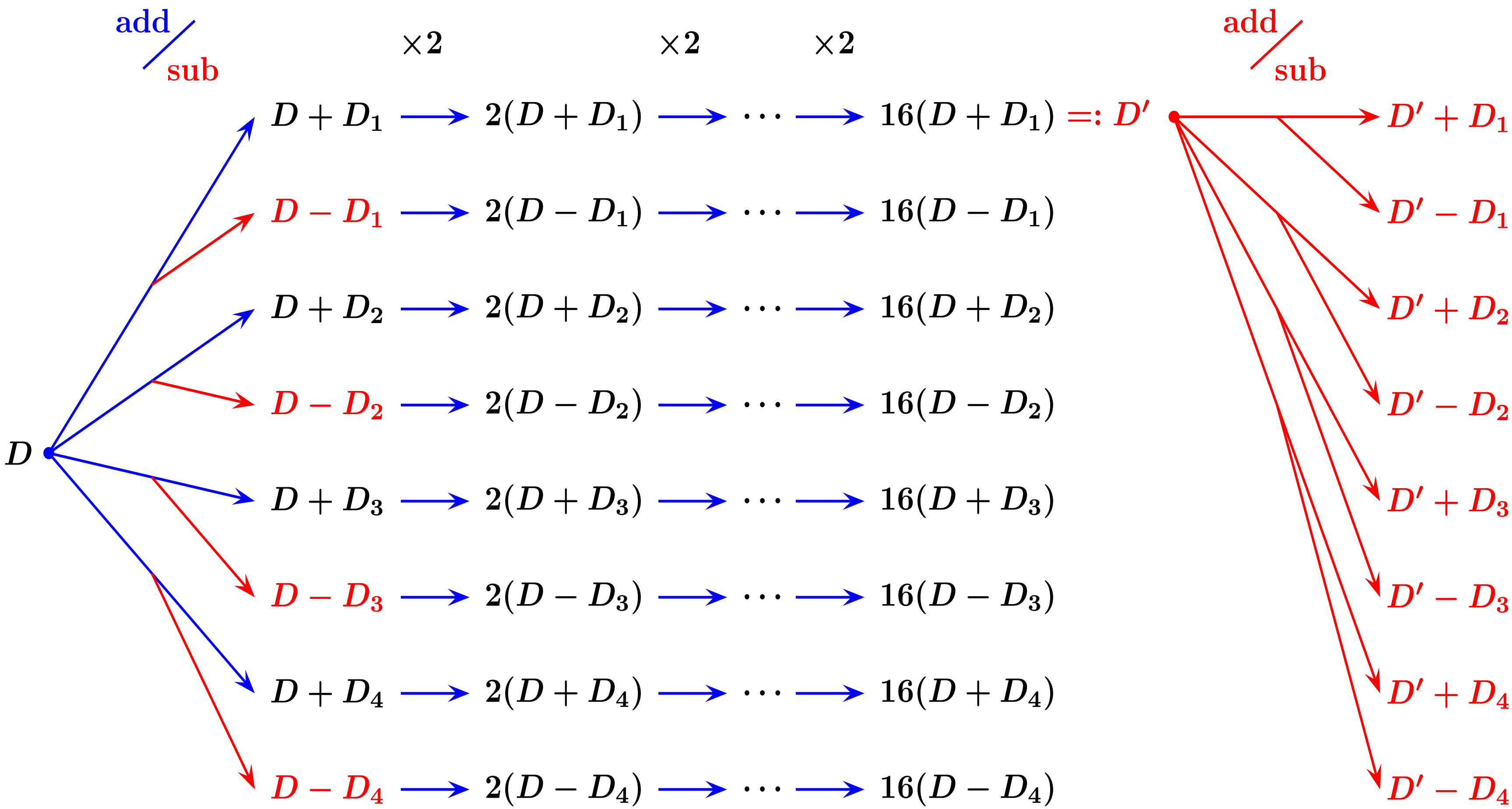

add
sub
×2
×2
×2
D
$D + D_1 \longrightarrow 2(D + D_1) \longrightarrow \cdots \longrightarrow 16(D + D_1)$
$D - D_1 \longrightarrow 2(D - D_1) \longrightarrow \cdots \longrightarrow 16(D - D_1)$
$D + D_2 \longrightarrow 2(D + D_2) \longrightarrow \cdots \longrightarrow 16(D + D_2)$
$D - D_2 \longrightarrow 2(D - D_2) \longrightarrow \cdots \longrightarrow 16(D - D_2)$
$D + D_3 \longrightarrow 2(D + D_3) \longrightarrow \cdots \longrightarrow 16(D + D_3)$
$D - D_3 \longrightarrow 2(D - D_3) \longrightarrow \cdots \longrightarrow 16(D - D_3)$
$D + D_4 \longrightarrow 2(D + D_4) \longrightarrow \cdots \longrightarrow 16(D + D_4) =: D'$
$D - D_4 \longrightarrow 2(D - D_4) \longrightarrow \cdots \longrightarrow 16(D - D_4)$
add
sub
$D' + D_1$
$D' - D_1$
$D' + D_2$
$D' - D_2$
$D' + D_3$
$D' - D_3$
$D' + D_4$
$D' - D_4$

Random Walk


add
sub
×2
×2
×2
D
D + D1 → 2(D + D1) → ··· → 16(D + D1)
D − D1 → 2(D − D1) → ··· → 16(D − D1)
D + D2 → 2(D + D2) → ··· → 16(D + D2) =: D′
D − D2 → 2(D − D2) → ··· → 16(D − D2)
D + D3 → 2(D + D3) → ··· → 16(D + D3)
D − D3 → 2(D − D3) → ··· → 16(D − D3)
D + D4 → 2(D + D4) → ··· → 16(D + D4)
D − D4 → 2(D − D4) → ··· → 16(D − D4)
add
sub
D′ + D1
D′ − D1
D′ + D2
D′ − D2
D′ + D3
D′ − D3
D′ + D4
D′ − D4


add / sub
×2
×2
×2
D
D + D1 → 2(D + D1) → ··· → 16(D + D1)
D − D1 → 2(D − D1) → ··· → 16(D − D1)
D + D2 → 2(D + D2) → ··· → 16(D + D2)
D − D2 → 2(D − D2) → ··· → 16(D − D2)
D + D3 → 2(D + D3) → ··· → 16(D + D3) =: D′
D − D3 → 2(D − D3) → ··· → 16(D − D3)
D + D4 → 2(D + D4) → ··· → 16(D + D4)
D − D4 → 2(D − D4) → ··· → 16(D − D4)
add / sub
D′ + D1
D′ − D1
D′ + D2
D′ − D2
D′ + D3
D′ − D3
D′ + D4
D′ − D4

Random Walk



Courtesy of Roberto Avanzi

Group Operation

- Once $D + D_i$ is computed, we can re-use some of the operations to compute $D - D_i$ (saves up to half of the cost of an addition).

Group Operation

- Once $D + D_i$ is computed, we can re-use some of the operations to compute $D - D_i$ (saves up to half of the cost of an addition).
- If doublings are cheaper than additions, we can do several blocks of doublings between each blocks of additions (we stopped at four).

Group Operation

- Once $D + D_i$ is computed, we can re-use some of the operations to compute $D - D_i$ (saves up to half of the cost of an addition).
- If doublings are cheaper than additions, we can do several blocks of doublings between each blocks of additions (we stopped at four).
- We can use Montgomery's trick for the inversions (and adapt our group operations).

Group Operation

- The D_i are selected wisely:
  - They are random linear combinations of D_a and D_b of the form $[u_i(x), v_i(x)]$ with $u_i(x)$ irreducible of degree $3 = g - 1$.

- The D_i are selected wisely:
  - They are random linear combinations of D_a and D_b of the form $[u_i(x), v_i(x)]$ with $u_i(x)$ irreducible of degree $3 = g - 1$.
  - A group addition when u_i has degree 3 costs about $2/3$ of a general group addition.
  - From $D + D1$, computing $D - D_1$ takes about $2/5$ of a general group addition.
  - Choosing u_i irreducible minimizes the risk of having $\gcd(u, u_i) \neq 1$.

Group Operation

- The D_i are selected wisely:
  - They are random linear combinations of D_a and D_b of the form $[u_i(x), v_i(x)]$ with $u_i(x)$ irreducible of degree $3 = g - 1$.
  - A group addition when u_i has degree 3 costs about $2/3$ of a general group addition.
  - From $D + D1$, computing $D - D_1$ takes about $2/5$ of a general group addition.
  - Choosing u_i irreducible minimizes the risk of having $\gcd(u, u_i) \neq 1$.
  - It takes $\approx 3q$ steps of pre-random-walk to find each D_i (goes into the $+\epsilon$).

Given a polynomial of small degree d over $\mathbb{F}_q$, the standard factorization algorithms take $O(d^2 \log_2(q))$ field operations to test if it splits, and to factor it into linear terms when possible.

For fields of characteristic two, we can improve on the standard methods.

Factorization

Given a polynomial of small degree d over $\mathbb{F}_q$, the standard factorization algorithms take $O(d^2 \log_2(q))$ field operations to test if it splits, and to factor it into linear terms when possible.

For fields of characteristic two, we can improve on the standard methods.

Taking advantage of the Frobenius over $\mathbb{F}_2$, we can get an algorithm which takes

$$O(d^3 \log_2(\log_2(q))) \quad \text{field operations}$$

with a nice constant in the $O(\ldots)$

Factorization

Given a polynomial of small degree d over $\mathbb{F}_q$, the standard factorization algorithms take $O(d^2 \log_2(q))$ field operations to test if it splits, and to factor it into linear terms when possible.

For fields of characteristic two, we can improve on the standard methods.

Taking advantage of the Frobenius over $\mathbb{F}_2$, we can get an algorithm which takes

$$O(d^3 \log_2(\log_2(q))) \quad \text{field operations}$$

with a nice constant in the $O(\ldots)$ (preprint in progress).

After several hours of fun, and many pages of operation counts and implementation (finding ratios between the different operations), we found

Field size	step of random walk	average group op.
53	≈ 178.187 M	≈ 50.139 M
73	≈ 194.688 M	≈ 49.813 M

After several hours of fun, and many pages of operation counts and implementation (finding ratios between the different operations), we found

Field size	step of random walk	average group op.
53	≈ 178.187 M	≈ 50.139 M
73	≈ 194.688 M	≈ 49.813 M

giving us

$$c_W(53) \approx 3.554 \qquad c_L(73) \approx 3.9084$$
$$c_L(53) \approx 0.1546 \qquad c_L(73) \approx 0.083125$$

After several hours of fun, and many pages of operation counts and implementation (finding ratios between the different operations), we found

Field size	step of random walk	average group op.
53	≈ 178.187 M	≈ 50.139 M
73	≈ 194.688 M	≈ 49.813 M

giving us

$$c_W(53) \approx 3.554 \qquad c_L(73) \approx 3.9084$$
$$c_L(53) \approx 0.1546 \qquad c_L(73) \approx 0.083125$$

Combining everything to get c_T in $T_{4,min} = c_T q^{14/9}$, we find

$$c_T(53) \approx 2.105 \qquad c_T(73) \approx 1.547$$

If we try to apply the same approach to 2LP (without the filter), we find for $c_{\tilde{T}}$ in $\tilde{T}_{min} = c_{\tilde{T}} q^{3/2}$:

$$c_{\tilde{T}}(53) \sim 9.133$$

$$c_{\tilde{T}}(73) \sim 8.524$$

Back to 2LP

If we try to apply the same approach to 2LP (without the filter), we find for $c_{\tilde{T}}$ in $\tilde{T}_{min} = c_{\tilde{T}} q^{3/2}$:

$$c_{\tilde{T}}(53) \sim 9.133$$

$$c_{\tilde{T}}(73) \sim 8.524$$

However, those numbers hide a few problems:

- The analysis uses some approximations which are too rough at these field sizes
- Some of the bounds may not be tight

Back to 2LP

If we try to apply the same approach to 2LP (without the filter), we find for $c_{\tilde{T}}$ in $\tilde{T}_{min} = c_{\tilde{T}} q^{3/2}$:

$$c_{\tilde{T}}(53) \sim 9.133$$

$$c_{\tilde{T}}(73) \sim 8.524$$

However, those numbers hide a few problems:

- The analysis uses some approximations which are too rough at these field sizes
- Some of the bounds may not be tight
- We are ignoring the cost of working in the graph of large primes

Comparison

At 53 bits, we find a total of $\approx 2^{83.518}$ average group operations for 1LP and $\sim 2^{82.691}$ for 2LP.

At 53 bits, we find a total of $\approx 2^{83.518}$ average group operations for 1LP and $\sim 2^{82.691}$ for 2LP.

This is closer to the security of an elliptic curve over a binary field of 166 to 168 bits than the 159 bits we would have expected from the $O(\ldots)$ (putting aside issues of Weil descent).

At 53 bits, we find a total of $\approx 2^{83.518}$ average group operations for 1LP and $\sim 2^{82.691}$ for 2LP.

This is closer to the security of an elliptic curve over a binary field of 166 to 168 bits than the 159 bits we would have expected from the $O(\ldots)$ (putting aside issues of Weil descent).

At 73 bits, we find a total of $\approx 2^{114.185}$ average group operations for 1LP and $\sim 2^{112.59}$ for 2LP.

We get a security level between $E(\mathbb{F}_{2^{226}})$ and $E(\mathbb{F}_{2^{229}})$ instead of $E(\mathbb{F}_{2^{219}})$ (from the O-notation of 2LP).

Comparison

At 53 bits, we find a total of $\approx 2^{83.518}$ average group operations for 1LP and $\sim 2^{82.691}$ for 2LP.

This is closer to the security of an elliptic curve over a binary field of 166 to 168 bits than the 159 bits we would have expected from the $O(\dots)$ (putting aside issues of Weil descent).

At 73 bits, we find a total of $\approx 2^{114.185}$ average group operations for 1LP and $\sim 2^{112.59}$ for 2LP.

We get a security level between $E(\mathbb{F}_{2^{226}})$ and $E(\mathbb{F}_{2^{229}})$ instead of $E(\mathbb{F}_{2^{219}})$ (from the O-notation of 2LP).

Ignoring the constants leads us to underestimate the security of hyperelliptic curve of genus four by a few bits.

Preliminary results for 2LP in genus 3 at 59 bits gives

$$\sim 7.903 \left(2^{59}\right)^{4/3} \sim 2^{81.649}$$

average group operations.

Genus 3

Preliminary results for 2LP in genus 3 at 59 bits gives

$$\sim 7.903\left(2^{59}\right)^{4/3} \sim 2^{81.649}$$

average group operations.

Note: This estimates suffers from similar issues as those for 2LP in genus 4, on top of which the computations have to be checked...