NTT

Classical Attacks on Elliptic Curve Cryptosystems

An Overview

Mehdi Tibouchi

NTT Secure Platform Laboratories

ECC Workshop 2018, Osaka University, 2018–11–17

Outline

Outline

Outline

Groups

A group G is a set together with some operation $*$ sending two elements a, b to another element $a = b * c$, such that:

1. $*$ is associative: $(a * b) * c = a * (b * c)$ for all a, b, c
2. there exists an identity element e: $e * a = a * e = a$ for all a
3. all elements are invertible: for all a, there exists a' such that $a * a' = a' * a = e$

Easy properties: e is unique; each element a has a unique inverse, denoted a^{-1}

Examples of groups

- The set $\mathbb{Z}$ of integers is a group under $+$
  - but not under $\times$!
- The set $\mathbb{Q}^*$ (resp. $\mathbb{R}^*$) of non-zero rational numbers (resp. real numbers) is a group under $\times$
  - we have to remove 0, which has no inverse
- For any integer n, the set $\{0, 1, \ldots, n-1\}$ under addition modulo n is a group, denoted $\mathbb{Z}/n\mathbb{Z}$
- The set $\mathbb{Z}/n\mathbb{Z} \setminus \{0\}$ is stable under multiplication modulo n if and only if $n = p$ is prime. In that case, it is a group, denoted $(\mathbb{Z}/p\mathbb{Z})^\times$ or $\mathbb{F}_p^*$.
- Elliptic curves are groups! (more about this later)
- In all these examples, the groups are commutative: for all a, b, $a * b = b * a$. There are also non-commutative groups, like groups of invertible matrices. Not so important for cryptography.

Examples of groups

- The set $\mathbb{Z}$ of integers is a group under $+$
  - but not under $\times$!
- The set $\mathbb{Q}^*$ (resp. $\mathbb{R}^*$) of non-zero rational numbers (resp. real numbers) is a group under $\times$
  - we have to remove 0, which has no inverse
- For any integer n, the set $\{0, 1, \ldots, n-1\}$ under addition modulo n is a group, denoted $\mathbb{Z}/n\mathbb{Z}$
- The set $\mathbb{Z}/n\mathbb{Z} \setminus \{0\}$ is stable under multiplication modulo n if and only if $n = p$ is prime. In that case, it is a group, denoted $(\mathbb{Z}/p\mathbb{Z})^\times$ or $\mathbb{F}_p^*$.
- Elliptic curves are groups! (more about this later)
- In all these examples, the groups are commutative: for all a, b, $a * b = b * a$. There are also non-commutative groups, like groups of invertible matrices. Not so important for cryptography.

6/39
©2018 NTT Secure Platform Laboratories

Examples of groups

- The set $\mathbb{Z}$ of integers is a group under $+$
  - but not under $\times$!
- The set $\mathbb{Q}^*$ (resp. $\mathbb{R}^*$) of non-zero rational numbers (resp. real numbers) is a group under $\times$
  - we have to remove 0, which has no inverse
- For any integer n, the set $\{0, 1, \ldots, n-1\}$ under addition modulo n is a group, denoted $\mathbb{Z}/n\mathbb{Z}$
- The set $\mathbb{Z}/n\mathbb{Z} \setminus \{0\}$ is stable under multiplication modulo n if and only if $n = p$ is prime. In that case, it is a group, denoted $(\mathbb{Z}/p\mathbb{Z})^\times$ or $\mathbb{F}_p^*$.
- Elliptic curves are groups! (more about this later)
- In all these examples, the groups are commutative: for all a, b, $a * b = b * a$. There are also non-commutative groups, like groups of invertible matrices. Not so important for cryptography.

6/39
©2018 NTT Secure Platform Laboratories

Examples of groups

- The set $\mathbb{Z}$ of integers is a group under $+$
  - but not under $\times$!
- The set $\mathbb{Q}^*$ (resp. $\mathbb{R}^*$) of non-zero rational numbers (resp. real numbers) is a group under $\times$
  - we have to remove 0, which has no inverse
- For any integer n, the set $\{0, 1, \ldots, n-1\}$ under addition modulo n is a group, denoted $\mathbb{Z}/n\mathbb{Z}$
- The set $\mathbb{Z}/n\mathbb{Z} \setminus \{0\}$ is stable under multiplication modulo n if and only if $n = p$ is prime. In that case, it is a group, denoted $(\mathbb{Z}/p\mathbb{Z})^\times$ or $\mathbb{F}_p^*$.
- Elliptic curves are groups! (more about this later)
- In all these examples, the groups are commutative: for all a, b, $a * b = b * a$. There are also non-commutative groups, like groups of invertible matrices. Not so important for cryptography.

Examples of groups

- The set $\mathbb{Z}$ of integers is a group under $+$
  - but not under $\times$!
- The set $\mathbb{Q}^*$ (resp. $\mathbb{R}^*$) of non-zero rational numbers (resp. real numbers) is a group under $\times$
  - we have to remove 0, which has no inverse
- For any integer n, the set $\{0, 1, \ldots, n-1\}$ under addition modulo n is a group, denoted $\mathbb{Z}/n\mathbb{Z}$
- The set $\mathbb{Z}/n\mathbb{Z} \setminus \{0\}$ is stable under multiplication modulo n if and only if $n = p$ is prime. In that case, it is a group, denoted $(\mathbb{Z}/p\mathbb{Z})^\times$ or $\mathbb{F}_p^*$.
- Elliptic curves are groups! (more about this later)
- In all these examples, the groups are commutative: for all a, b, $a * b = b * a$. There are also non-commutative groups, like groups of invertible matrices. Not so important for cryptography.

6/39
©2018 NTT Secure Platform Laboratories

Examples of groups

- The set $\mathbb{Z}$ of integers is a group under $+$
  - but not under $\times$!
- The set $\mathbb{Q}^*$ (resp. $\mathbb{R}^*$) of non-zero rational numbers (resp. real numbers) is a group under $\times$
  - we have to remove 0, which has no inverse
- For any integer n, the set $\{0, 1, \ldots, n-1\}$ under addition modulo n is a group, denoted $\mathbb{Z}/n\mathbb{Z}$
- The set $\mathbb{Z}/n\mathbb{Z} \setminus \{0\}$ is stable under multiplication modulo n if and only if $n = p$ is prime. In that case, it is a group, denoted $(\mathbb{Z}/p\mathbb{Z})^\times$ or $\mathbb{F}_p^*$.
- Elliptic curves are groups! (more about this later)
- In all these examples, the groups are commutative: for all a, b, $a * b = b * a$. There are also non-commutative groups, like groups of invertible matrices. Not so important for cryptography.

6/39
©2018 NTT Secure Platform Laboratories

Finite groups

- From now on, we restrict attention to finite groups (i.e. the underlying set G is finite). The number N of elements is also called the order of the group
- For now, we denote the group operation multiplicatively: $a * b$ is just $a \cdot b$ or ab, and the identity is 1. We will switch to additive notation later
- Accordingly, for any positive integer m and any group element g, we denote by g^m the group element $g \cdot g \cdots g$ (m times). We also let $g^0 = 1$ and $g^{-m} = (g^{-1})^m$

Finite groups

- From now on, we restrict attention to finite groups (i.e. the underlying set G is finite). The number N of elements is also called the order of the group
- For now, we denote the group operation multiplicatively: $a * b$ is just $a \cdot b$ or ab, and the identity is 1. We will switch to additive notation later
- Accordingly, for any positive integer m and any group element g, we denote by g^m the group element $g \cdot g \cdots g$ (m times). We also let $g^0 = 1$ and $g^{-m} = (g^{-1})^m$

Finite groups

- From now on, we restrict attention to finite groups (i.e. the underlying set G is finite). The number N of elements is also called the order of the group
- For now, we denote the group operation multiplicatively: $a * b$ is just $a \cdot b$ or ab, and the identity is 1. We will switch to additive notation later
- Accordingly, for any positive integer m and any group element g, we denote by g^m the group element $g \cdot g \cdots g$ (m times). We also let $g^0 = 1$ and $g^{-m} = (g^{-1})^m$

7/39
©2018 NTT Secure Platform Laboratories

Orders of elements, cyclic groups

- For each g in a finite group, there is a smallest positive integer m such that $g^m = 1$. This is the order of g
  - indeed, by finiteness, there exists $n > n' > 0$ such that $g^n = g^{n'}$; then $g^{n-n'} = 1$. This shows the existence of an $m > 0$ such that $g^m = 1$; just take the smallest
  - the order m of g always divides the order N of the entire group (Legendre)
- For a fixed g, the set of all elements of the form g^m is stable under the group law and under inversion, and it contains 1: it is a subgroup of G denoted by $\langle g \rangle$
- If $G = \langle g \rangle$, we say that the group G is cyclic and g is a generator of G
  - a cyclic group is clearly always commutative!

Orders of elements, cyclic groups

- For each g in a finite group, there is a smallest positive integer m such that $g^m = 1$. This is the order of g
  - indeed, by finiteness, there exists $n > n' > 0$ such that $g^n = g^{n'}$; then $g^{n-n'} = 1$. This shows the existence of an $m > 0$ such that $g^m = 1$; just take the smallest
  - the order m of g always divides the order N of the entire group (Legendre)
- For a fixed g, the set of all elements of the form g^m is stable under the group law and under inversion, and it contains 1: it is a subgroup of G denoted by $\langle g \rangle$
- If $G = \langle g \rangle$, we say that the group G is cyclic and g is a generator of G
  - a cyclic group is clearly always commutative!

8/39
©2018 NTT Secure Platform Laboratories

Orders of elements, cyclic groups

- For each g in a finite group, there is a smallest positive integer m such that $g^m = 1$. This is the order of g
  - indeed, by finiteness, there exists $n > n' > 0$ such that $g^n = g^{n'}$; then $g^{n-n'} = 1$. This shows the existence of an $m > 0$ such that $g^m = 1$; just take the smallest
  - the order m of g always divides the order N of the entire group (Legendre)
- For a fixed g, the set of all elements of the form g^m is stable under the group law and under inversion, and it contains 1: it is a subgroup of G denoted by $\langle g \rangle$
- If $G = \langle g \rangle$, we say that the group G is cyclic and g is a generator of G
  - a cyclic group is clearly always commutative!

The discrete logarithm problem

- Let G be a cyclic group of order N with generator g. By definition, for all $h \in G$, there exists some integer x such that $h = g^x$.
- By analogy with real logarithms, x is called the discrete logarithm of h wrt g
  - x is unique up to addition of a multiple of N (i.e. unique modulo N)
- The discrete logarithm problem is the problem of computing x given G, N, g, h
- This assumes that we can compute in G: there exist concrete representations of the elements of G as bit strings, and efficient algorithms to compute the group law in G and inversion. We assume that from now on

The discrete logarithm problem

- Let G be a cyclic group of order N with generator g. By definition, for all $h \in G$, there exists some integer x such that $h = g^x$.
- By analogy with real logarithms, x is called the discrete logarithm of h wrt g
  - x is unique up to addition of a multiple of N (i.e. unique modulo N)
- The discrete logarithm problem is the problem of computing x given G, N, g, h
- This assumes that we can compute in G: there exist concrete representations of the elements of G as bit strings, and efficient algorithms to compute the group law in G and inversion. We assume that from now on

9/39
©2018 NTT Secure Platform Laboratories

The discrete logarithm problem

- Let G be a cyclic group of order N with generator g. By definition, for all $h \in G$, there exists some integer x such that $h = g^x$.
- By analogy with real logarithms, x is called the discrete logarithm of h wrt g
  - x is unique up to addition of a multiple of N (i.e. unique modulo N)
- The discrete logarithm problem is the problem of computing x given G, N, g, h
- This assumes that we can compute in G: there exist concrete representations of the elements of G as bit strings, and efficient algorithms to compute the group law in G and inversion. We assume that from now on

9/39
©2018 NTT Secure Platform Laboratories

The discrete logarithm problem

- Let G be a cyclic group of order N with generator g. By definition, for all $h \in G$, there exists some integer x such that $h = g^x$.
- By analogy with real logarithms, x is called the discrete logarithm of h wrt g
  - x is unique up to addition of a multiple of N (i.e. unique modulo N)
- The discrete logarithm problem is the problem of computing x given G, N, g, h
- This assumes that we can compute in G: there exist concrete representations of the elements of G as bit strings, and efficient algorithms to compute the group law in G and inversion. We assume that from now on

Outline

Solving the discrete logarithm problem

- The hardness of the discrete logarithm problem (DLP) is at the core of the security arguments for all cryptography based on groups, incl. elliptic curve cryptography
- Hence, if we can solve DLP efficiently, we can break all group-based crypto
- Various approaches to attack the DLP depending on the group we consider
- However, arguably the most important approaches are those that are generic: work in all group, independently of the particular group law or representation of group elements
- We now discuss those approaches

Solving the discrete logarithm problem

- The hardness of the discrete logarithm problem (DLP) is at the core of the security arguments for all cryptography based on groups, incl. elliptic curve cryptography
- Hence, if we can solve DLP efficiently, we can break all group-based crypto
- Various approaches to attack the DLP depending on the group we consider
- However, arguably the most important approaches are those that are generic: work in all group, independently of the particular group law or representation of group elements
- We now discuss those approaches

Solving the discrete logarithm problem

- The hardness of the discrete logarithm problem (DLP) is at the core of the security arguments for all cryptography based on groups, incl. elliptic curve cryptography
- Hence, if we can solve DLP efficiently, we can break all group-based crypto
- Various approaches to attack the DLP depending on the group we consider
- However, arguably the most important approaches are those that are generic: work in all group, independently of the particular group law or representation of group elements
- We now discuss those approaches

Solving the discrete logarithm problem

- The hardness of the discrete logarithm problem (DLP) is at the core of the security arguments for all cryptography based on groups, incl. elliptic curve cryptography
- Hence, if we can solve DLP efficiently, we can break all group-based crypto
- Various approaches to attack the DLP depending on the group we consider
- However, arguably the most important approaches are those that are generic: work in all group, independently of the particular group law or representation of group elements
- We now discuss those approaches

11/39 ©2018 NTT Secure Platform Laboratories

Solving the discrete logarithm problem

- The hardness of the discrete logarithm problem (DLP) is at the core of the security arguments for all cryptography based on groups, incl. elliptic curve cryptography
- Hence, if we can solve DLP efficiently, we can break all group-based crypto
- Various approaches to attack the DLP depending on the group we consider
- However, arguably the most important approaches are those that are generic: work in all group, independently of the particular group law or representation of group elements
- We now discuss those approaches

11/39 ©2018 NTT Secure Platform Laboratories

A trivial discrete log algorithm

- Here is a trivial algorithm for the DLP
- Given g and h, the goal is to find x such that $h = g^x$
- Simple way:
  1. $x \leftarrow 0$
  2. if $h = g^x$, we are done
  3. otherwise, $x \leftarrow x + 1$ and try again
- We know for sure that this algorithm will find a solution. Since there is such a solution x such that $0 \leq x < N$, time complexity is $O(N)$ in the worst case (and space complexity is $O(1)$)
- Note that this algorithm is exponential in the bit size of the group (which is $\log_2 N$)

A trivial discrete log algorithm

- Here is a trivial algorithm for the DLP
- Given g and h, the goal is to find x such that $h = g^x$
- Simple way:
  1. $x \leftarrow 0$
  2. if $h = g^x$, we are done
  3. otherwise, $x \leftarrow x + 1$ and try again
- We know for sure that this algorithm will find a solution. Since there is such a solution x such that $0 \leq x < N$, time complexity is $O(N)$ in the worst case (and space complexity is $O(1)$)
- Note that this algorithm is exponential in the bit size of the group (which is $\log_2 N$)

A trivial discrete log algorithm

- Here is a trivial algorithm for the DLP
- Given g and h, the goal is to find x such that $h = g^x$
- Simple way:
  1. $x \leftarrow 0$
  2. if $h = g^x$, we are done
  3. otherwise, $x \leftarrow x + 1$ and try again
- We know for sure that this algorithm will find a solution. Since there is such a solution x such that $0 \leq x < N$, time complexity is $O(N)$ in the worst case (and space complexity is $O(1)$)
- Note that this algorithm is exponential in the bit size of the group (which is $\log_2 N$)

12/39
©2018 NTT Secure Platform Laboratories

A trivial discrete log algorithm

- Here is a trivial algorithm for the DLP
- Given g and h, the goal is to find x such that $h = g^x$
- Simple way:
  1. $x \leftarrow 0$
  2. if $h = g^x$, we are done
  3. otherwise, $x \leftarrow x + 1$ and try again
- We know for sure that this algorithm will find a solution. Since there is such a solution x such that $0 \leq x < N$, time complexity is $O(N)$ in the worst case (and space complexity is $O(1)$)
- Note that this algorithm is exponential in the bit size of the group (which is $\log_2 N$)

12/39
©2018 NTT Secure Platform Laboratories

A trivial discrete log algorithm

- Here is a trivial algorithm for the DLP
- Given g and h, the goal is to find x such that $h = g^x$
- Simple way:
  1. $x \leftarrow 0$
  2. if $h = g^x$, we are done
  3. otherwise, $x \leftarrow x + 1$ and try again
- We know for sure that this algorithm will find a solution. Since there is such a solution x such that $0 \leq x < N$, time complexity is $O(N)$ in the worst case (and space complexity is $O(1)$)
- Note that this algorithm is exponential in the bit size of the group (which is $\log_2 N$)

12/39
©2018 NTT Secure Platform Laboratories

A trivial discrete log algorithm

- Here is a trivial algorithm for the DLP
- Given g and h, the goal is to find x such that $h = g^x$
- Simple way:
  1. $x \leftarrow 0$
  2. if $h = g^x$, we are done
  3. otherwise, $x \leftarrow x + 1$ and try again
- We know for sure that this algorithm will find a solution. Since there is such a solution x such that $0 \leq x < N$, time complexity is $O(N)$ in the worst case (and space complexity is $O(1)$)
- Note that this algorithm is exponential in the bit size of the group (which is $\log_2 N$)

A trivial discrete log algorithm

- Here is a trivial algorithm for the DLP
- Given g and h, the goal is to find x such that $h = g^x$
- Simple way:
  1. $x \leftarrow 0$
  2. if $h = g^x$, we are done
  3. otherwise, $x \leftarrow x + 1$ and try again
- We know for sure that this algorithm will find a solution. Since there is such a solution x such that $0 \leq x < N$, time complexity is $O(N)$ in the worst case (and space complexity is $O(1)$)
- Note that this algorithm is exponential in the bit size of the group (which is $\log_2 N$)

A trivial discrete log algorithm

- Here is a trivial algorithm for the DLP
- Given g and h, the goal is to find x such that $h = g^x$
- Simple way:
  1. $x \leftarrow 0$
  2. if $h = g^x$, we are done
  3. otherwise, $x \leftarrow x + 1$ and try again
- We know for sure that this algorithm will find a solution. Since there is such a solution x such that $0 \leq x < N$, time complexity is $O(N)$ in the worst case (and space complexity is $O(1)$)
- Note that this algorithm is exponential in the bit size of the group (which is $\log_2 N$)

Baby step, giant step (I)

- We can do much better with some memory: the baby step, giant step (BSGS) algorithm
- Let $m = \lceil \sqrt{N} \rceil$. One can write the discrete log x of h as $x = y + mz$, with $0 \leq y, z < m$. This gives:

$$h = g^{y+mz} = g^y \cdot g^{mz} \quad \text{hence} \quad h \cdot g^{-y} = g^{mz}$$

- In time $\widetilde{O}(\sqrt{N})$ and space $O(\sqrt{N})$, construct the following list and sort it (so that we can search through it efficiently):

$$L = \{g^0, g^m, g^{m \cdot 2}, \cdots, g^{m \cdot (m-1)}\}$$

By construction, g^{mz} is in the list

Baby step, giant step (I)

- We can do much better with some memory: the baby step, giant step (BSGS) algorithm
- Let $m = \lceil \sqrt{N} \rceil$. One can write the discrete log x of h as $x = y + mz$, with $0 \le y, z < m$. This gives:

$$h = g^{y+mz} = g^y \cdot g^{mz} \quad \text{hence} \quad h \cdot g^{-y} = g^{mz}$$

- In time $\widetilde{O}(\sqrt{N})$ and space $O(\sqrt{N})$, construct the following list and sort it (so that we can search through it efficiently):

$$L = \{g^0, g^m, g^{m \cdot 2}, \cdots, g^{m \cdot (m-1)}\}$$

By construction, g^{mz} is in the list

Baby step, giant step (I)

- We can do much better with some memory: the baby step, giant step (BSGS) algorithm
- Let $m = \lceil \sqrt{N} \rceil$. One can write the discrete log x of h as $x = y + mz$, with $0 \le y, z < m$. This gives:

$$h = g^{y+mz} = g^y \cdot g^{mz} \quad \text{hence} \quad h \cdot g^{-y} = g^{mz}$$

- In time $\widetilde{O}(\sqrt{N})$ and space $O(\sqrt{N})$, construct the following list and sort it (so that we can search through it efficiently):

$$L = \{g^0, g^m, g^{m \cdot 2}, \cdots, g^{m \cdot (m-1)}\}$$

By construction, g^{mz} is in the list

Baby step, giant step (II)

- Now do a similar search as before:
  1. $y \leftarrow 0$
  2. search for $h \cdot g^{-y}$ in the list L
  3. if it is found as g^{mz}, return $x = y + mz$ as the discrete log
  4. otherwise, $y \leftarrow y + 1$ and try again
- Total time complexity is $\widetilde{O}(\sqrt{N})$ and space complexity is $O(\sqrt{N})$. Still exponential, but exponentially faster than the trivial approach
- In practice the space complexity is usually prohibitive: e.g. if $N \approx 2^{128}$, the 2^{64} time complexity is manageable, but 2^{64} space is not!
- By choosing a different value for m, we can obtain different time-memory trade-offs. Not necessary though: same time complexity is achievable without memory!

Baby step, giant step (II)

- Now do a similar search as before:
  1. $y \leftarrow 0$
  2. search for $h \cdot g^{-y}$ in the list L
  3. if it is found as g^{mz}, return $x = y + mz$ as the discrete log
  4. otherwise, $y \leftarrow y + 1$ and try again
- Total time complexity is $\widetilde{O}(\sqrt{N})$ and space complexity is $O(\sqrt{N})$. Still exponential, but exponentially faster than the trivial approach
- In practice the space complexity is usually prohibitive: e.g. if $N \approx 2^{128}$, the 2^{64} time complexity is manageable, but 2^{64} space is not!
- By choosing a different value for m, we can obtain different time-memory trade-offs. Not necessary though: same time complexity is achievable without memory!

Baby step, giant step (II)

- Now do a similar search as before:
  1. $y \leftarrow 0$
  2. search for $h \cdot g^{-y}$ in the list L
  3. if it is found as g^{mz}, return $x = y + mz$ as the discrete log
  4. otherwise, $y \leftarrow y + 1$ and try again
- Total time complexity is $\widetilde{O}(\sqrt{N})$ and space complexity is $O(\sqrt{N})$. Still exponential, but exponentially faster than the trivial approach
- In practice the space complexity is usually prohibitive: e.g. if $N \approx 2^{128}$, the 2^{64} time complexity is manageable, but 2^{64} space is not!
- By choosing a different value for m, we can obtain different time-memory trade-offs. Not necessary though: same time complexity is achievable without memory!

Baby step, giant step (II)

- Now do a similar search as before:
  1. $y \leftarrow 0$
  2. search for $h \cdot g^{-y}$ in the list L
  3. if it is found as g^{mz}, return $x = y + mz$ as the discrete log
  4. otherwise, $y \leftarrow y + 1$ and try again
- Total time complexity is $\widetilde{O}(\sqrt{N})$ and space complexity is $O(\sqrt{N})$. Still exponential, but exponentially faster than the trivial approach
- In practice the space complexity is usually prohibitive: e.g. if $N \approx 2^{128}$, the 2^{64} time complexity is manageable, but 2^{64} space is not!
- By choosing a different value for m, we can obtain different time-memory trade-offs. Not necessary though: same time complexity is achievable without memory!

14/39
©2018 NTT Secure Platform Laboratories

Baby step, giant step (II)

- Now do a similar search as before:
  1. $y \leftarrow 0$
  2. search for $h \cdot g^{-y}$ in the list L
  3. if it is found as g^{mz}, return $x = y + mz$ as the discrete log
  4. otherwise, $y \leftarrow y + 1$ and try again
- Total time complexity is $\widetilde{O}(\sqrt{N})$ and space complexity is $O(\sqrt{N})$. Still exponential, but exponentially faster than the trivial approach
- In practice the space complexity is usually prohibitive: e.g. if $N \approx 2^{128}$, the 2^{64} time complexity is manageable, but 2^{64} space is not!
- By choosing a different value for m, we can obtain different time-memory trade-offs. Not necessary though: same time complexity is achievable without memory!

14/39
©2018 NTT Secure Platform Laboratories

Baby step, giant step (II)

- Now do a similar search as before:
  1. $y \leftarrow 0$
  2. search for $h \cdot g^{-y}$ in the list L
  3. if it is found as g^{mz}, return $x = y + mz$ as the discrete log
  4. otherwise, $y \leftarrow y + 1$ and try again
- Total time complexity is $\widetilde{O}(\sqrt{N})$ and space complexity is $O(\sqrt{N})$. Still exponential, but exponentially faster than the trivial approach
- In practice the space complexity is usually prohibitive: e.g. if $N \approx 2^{128}$, the 2^{64} time complexity is manageable, but 2^{64} space is not!
- By choosing a different value for m, we can obtain different time-memory trade-offs. Not necessary though: same time complexity is achievable without memory!

14/39
©2018 NTT Secure Platform Laboratories

Baby step, giant step (II)

- Now do a similar search as before:
  1. $y \leftarrow 0$
  2. search for $h \cdot g^{-y}$ in the list L
  3. if it is found as g^{mz}, return $x = y + mz$ as the discrete log
  4. otherwise, $y \leftarrow y + 1$ and try again
- Total time complexity is $\widetilde{O}(\sqrt{N})$ and space complexity is $O(\sqrt{N})$. Still exponential, but exponentially faster than the trivial approach
- In practice the space complexity is usually prohibitive: e.g. if $N \approx 2^{128}$, the 2^{64} time complexity is manageable, but 2^{64} space is not!
- By choosing a different value for m, we can obtain different time-memory trade-offs. Not necessary though: same time complexity is achievable without memory!

Baby step, giant step (II)

- Now do a similar search as before:
  1. $y \leftarrow 0$
  2. search for $h \cdot g^{-y}$ in the list L
  3. if it is found as g^{mz}, return $x = y + mz$ as the discrete log
  4. otherwise, $y \leftarrow y + 1$ and try again
- Total time complexity is $\widetilde{O}(\sqrt{N})$ and space complexity is $O(\sqrt{N})$. Still exponential, but exponentially faster than the trivial approach
- In practice the space complexity is usually prohibitive: e.g. if $N \approx 2^{128}$, the 2^{64} time complexity is manageable, but 2^{64} space is not!
- By choosing a different value for m, we can obtain different time-memory trade-offs. Not necessary though: same time complexity is achievable without memory!

Pollard's rho algorithm (I)

- The best generic algorithm for discrete logarithms is Pollard's rho: it uses $O(\sqrt{N})$ time and constant space.
- Basic ingredient: cycle-finding for random functions. If $f: X \to X$ is a random function of a set of cardinality N to itself, and we iterate f on a random element x_0:

$$x_1 = f(x_0); \quad x_2 = f(x_1); \cdots$$

then there are integers $s, t = O(\sqrt{N})$ whp such that $x_t = x_{t+s}$. A well-known algorithm (Floyd) then allows to find s, t in time $O(\sqrt{N})$ and constant memory

Pollard's rho algorithm (I)

- The best generic algorithm for discrete logarithms is Pollard's rho: it uses $O(\sqrt{N})$ time and constant space.
- Basic ingredient: cycle-finding for random functions. If $f: X \to X$ is a random function of a set of cardinality N to itself, and we iterate f on a random element x_0:

$$x_1 = f(x_0); \quad x_2 = f(x_1); \cdots$$

then there are integers $s, t = O(\sqrt{N})$ whp such that $x_t = x_{t+s}$. A well-known algorithm (Floyd) then allows to find s, t in time $O(\sqrt{N})$ and constant memory

Pollard's rho algorithm (II)

- Now consider the set $X = G$, and fix a random element $x_0 = g^{a_0} \cdot h^{b_0}$. Moreover, we construct f in such a way that f acts on each element by multiplication by some known powers of g and h. As a result, for all i, we know a_i, b_i such that

$$x_i = f^i(x_0) = g^{a_i} \cdot h^{b_i}$$

- Apply cycle-finding. In time $O(\sqrt{N})$ and constant space, we find whp integers a, b, a', b' such that:

$$g^a \cdot h^b = x_t = x_{t+s} = g^{a'} \cdot h^{b'}$$

Then, if $b - b'$ is coprime to N (happens with good probability), we deduce:

$$h = g^\alpha \quad \text{where} \quad \alpha \equiv -\frac{a - a'}{b - b'} \pmod{N}$$

Pollard's rho algorithm (II)

- Now consider the set $X = G$, and fix a random element $x_0 = g^{a_0} \cdot h^{b_0}$. Moreover, we construct f in such a way that f acts on each element by multiplication by some known powers of g and h. As a result, for all i, we know a_i, b_i such that

$$x_i = f^i(x_0) = g^{a_i} \cdot h^{b_i}$$

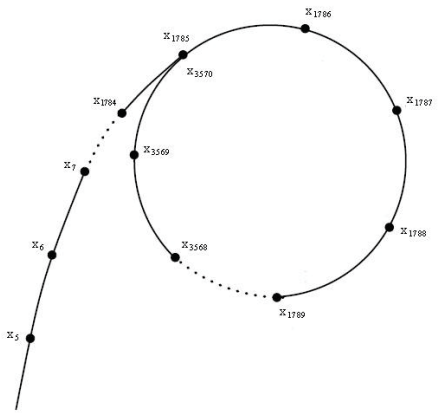
- Apply cycle-finding. In time $O(\sqrt{N})$ and constant space, we find whp integers a, b, a', b' such that:

$$g^a \cdot h^b = x_t = x_{t+s} = g^{a'} \cdot h^{b'}$$

Then, if $b - b'$ is coprime to N (happens with good probability), we deduce:

$$h = g^\alpha \quad \text{where} \quad \alpha \equiv -\frac{a - a'}{b - b'} \pmod{N}$$

Pollard's rho algorithm



Picture of the ρ in Pollard's rho (from Wikimedia Commons)

Pohlig–Hellman (I)

- There is one last thing we can do to speed up generic discrete logarithm computations: use the factorization of the group order N
- Suppose N factors as a product $N_1 N_2$ of two coprime integers. Then for a generator g of G, g^{N_2} is of order N_1 and g^{N_1} is of order N_2.
- We look for x such that $h = g^x$. Raising both sides to the power N_2, we get: $h^{N_2} = (g^{N_2})^x$, DLP in the group $\langle g^{N_2} \rangle$ of order N_1.
- Solve this DLP with Pollard's rho to obtain $x \bmod N_1$. Similarly, solve the DLP between g^{N_1} and h^{N_1} to find $x \bmod N_2$. Then apply the Chinese remainder theorem (CRT) to compute x
  - time complexity $O(\sqrt{N_1} + \sqrt{N_2})$ and constant space

Pohlig–Hellman (I)

- There is one last thing we can do to speed up generic discrete logarithm computations: use the factorization of the group order N
- Suppose N factors as a product $N_1 N_2$ of two coprime integers. Then for a generator g of G, g^{N_2} is of order N_1 and g^{N_1} is of order N_2.
- We look for x such that $h = g^x$. Raising both sides to the power N_2, we get: $h^{N_2} = (g^{N_2})^x$, DLP in the group $\langle g^{N_2} \rangle$ of order N_1.
- Solve this DLP with Pollard's rho to obtain $x \bmod N_1$. Similarly, solve the DLP between g^{N_1} and h^{N_1} to find $x \bmod N_2$. Then apply the Chinese remainder theorem (CRT) to compute x
  - time complexity $O(\sqrt{N_1} + \sqrt{N_2})$ and constant space

Pohlig–Hellman (I)

- There is one last thing we can do to speed up generic discrete logarithm computations: use the factorization of the group order N
- Suppose N factors as a product $N_1 N_2$ of two coprime integers. Then for a generator g of G, g^{N_2} is of order N_1 and g^{N_1} is of order N_2.
- We look for x such that $h = g^x$. Raising both sides to the power N_2, we get: $h^{N_2} = (g^{N_2})^x$, DLP in the group $\langle g^{N_2} \rangle$ of order N_1.
- Solve this DLP with Pollard's rho to obtain $x \bmod N_1$. Similarly, solve the DLP between g^{N_1} and h^{N_1} to find $x \bmod N_2$. Then apply the Chinese remainder theorem (CRT) to compute x
  - time complexity $O(\sqrt{N_1} + \sqrt{N_2})$ and constant space

Pohlig–Hellman (I)

- There is one last thing we can do to speed up generic discrete logarithm computations: use the factorization of the group order N
- Suppose N factors as a product $N_1 N_2$ of two coprime integers. Then for a generator g of G, g^{N_2} is of order N_1 and g^{N_1} is of order N_2.
- We look for x such that $h = g^x$. Raising both sides to the power N_2, we get: $h^{N_2} = (g^{N_2})^x$, DLP in the group $\langle g^{N_2} \rangle$ of order N_1.
- Solve this DLP with Pollard's rho to obtain $x \bmod N_1$. Similarly, solve the DLP between g^{N_1} and h^{N_1} to find $x \bmod N_2$. Then apply the Chinese remainder theorem (CRT) to compute x
  - time complexity $O(\sqrt{N_1} + \sqrt{N_2})$ and constant space

Pohlig–Hellman (II)

- More generally, if $N = p_1^{e_1} \cdots p_r^{e_r}$ is the prime factorization of N, that decomposition approach reduces the DLP in G to DLPs in groups of order $p_1^{e_1}, \ldots, p_r^{e_r}$.
- Then, a similar trick reduces the DLP in a group of order p^e to e instances of DLPs in groups of order p (relying on Pollard's kangaroo)
- Conclusion: can solve discrete logs in G in time $O(e_1\sqrt{p_1} + \cdots + e_r\sqrt{p_r})$ and polynomial space.
- For most N, this simply reduces to $O(\sqrt{p})$ time, p largest prime factor

Pohlig–Hellman (II)

- More generally, if $N = p_1^{e_1} \cdots p_r^{e_r}$ is the prime factorization of N, that decomposition approach reduces the DLP in G to DLPs in groups of order $p_1^{e_1}, \ldots, p_r^{e_r}$.
- Then, a similar trick reduces the DLP in a group of order p^e to e instances of DLPs in groups of order p (relying on Pollard's kangaroo)
- Conclusion: can solve discrete logs in G in time $O(e_1\sqrt{p_1} + \cdots + e_r\sqrt{p_r})$ and polynomial space.
- For most N, this simply reduces to $O(\sqrt{p})$ time, p largest prime factor

Pohlig–Hellman (II)

- More generally, if $N = p_1^{e_1} \cdots p_r^{e_r}$ is the prime factorization of N, that decomposition approach reduces the DLP in G to DLPs in groups of order $p_1^{e_1}, \ldots, p_r^{e_r}$.
- Then, a similar trick reduces the DLP in a group of order p^e to e instances of DLPs in groups of order p (relying on Pollard's kangaroo)
- Conclusion: can solve discrete logs in G in time $O(e_1\sqrt{p_1} + \cdots + e_r\sqrt{p_r})$ and polynomial space.
- For most N, this simply reduces to $O(\sqrt{p})$ time, p largest prime factor

Pohlig–Hellman (II)

- More generally, if $N = p_1^{e_1} \cdots p_r^{e_r}$ is the prime factorization of N, that decomposition approach reduces the DLP in G to DLPs in groups of order $p_1^{e_1}, \ldots, p_r^{e_r}$.
- Then, a similar trick reduces the DLP in a group of order p^e to e instances of DLPs in groups of order p (relying on Pollard's kangaroo)
- Conclusion: can solve discrete logs in G in time $O(e_1\sqrt{p_1} + \cdots + e_r\sqrt{p_r})$ and polynomial space.
- For most N, this simply reduces to $O(\sqrt{p})$ time, p largest prime factor

Shoup's lower bound

- Shoup proved that a generic algorithm for the DLP in a cyclic group of prime order p had to carry out $\Omega(\sqrt{p})$ group operations
- Therefore, among generic algorithms, Pollard's rho is optimal (up to a constant) for groups of prime order, and Pollard's rho + Pohlig–Hellman optimal in general
- On most elliptic curves, no better algorithm than those is known for the discrete log!
- Caveat: this is only about attacks on classical computers. Quantum computers generically break the DLP in polynomial time with Shor's algorithm

Shoup's lower bound

- Shoup proved that a generic algorithm for the DLP in a cyclic group of prime order p had to carry out $\Omega(\sqrt{p})$ group operations
- Therefore, among generic algorithms, Pollard's rho is optimal (up to a constant) for groups of prime order, and Pollard's rho + Pohlig–Hellman optimal in general
- On most elliptic curves, no better algorithm than those is known for the discrete log!
- Caveat: this is only about attacks on classical computers. Quantum computers generically break the DLP in polynomial time with Shor's algorithm

Shoup's lower bound

- Shoup proved that a generic algorithm for the DLP in a cyclic group of prime order p had to carry out $\Omega(\sqrt{p})$ group operations
- Therefore, among generic algorithms, Pollard's rho is optimal (up to a constant) for groups of prime order, and Pollard's rho + Pohlig–Hellman optimal in general
- On most elliptic curves, no better algorithm than those is known for the discrete log!
- Caveat: this is only about attacks on classical computers. Quantum computers generically break the DLP in polynomial time with Shor's algorithm

Shoup's lower bound

- Shoup proved that a generic algorithm for the DLP in a cyclic group of prime order p had to carry out $\Omega(\sqrt{p})$ group operations
- Therefore, among generic algorithms, Pollard's rho is optimal (up to a constant) for groups of prime order, and Pollard's rho + Pohlig–Hellman optimal in general
- On most elliptic curves, no better algorithm than those is known for the discrete log!
- Caveat: this is only about attacks on classical computers. Quantum computers generically break the DLP in polynomial time with Shor's algorithm

Outline

Additive group DLP

- There are of course some groups when the DLP can be solved much faster than by generic techniques
- Example: the cyclic group $\mathbb{Z}/N\mathbb{Z}$ of integers modulo N under addition
- This is an additive group. The DLP is to find, given two elements $a, b \in \mathbb{Z}/N\mathbb{Z}$, a value x such that $b \equiv ax \pmod{N}$
- This is just a division! Simply compute x as:

$$x \equiv \frac{b}{a} \pmod{N}$$

(computation done using the extended Euclidean algorithm)
- In $\mathbb{Z}/N\mathbb{Z}$, DLP can be solved in polynomial time (in $\log N$), even though generic algorithms are all exponential

Additive group DLP

- There are of course some groups when the DLP can be solved much faster than by generic techniques
- Example: the cyclic group $\mathbb{Z}/N\mathbb{Z}$ of integers modulo N under addition
- This is an additive group. The DLP is to find, given two elements $a, b \in \mathbb{Z}/N\mathbb{Z}$, a value x such that $b \equiv ax \pmod{N}$
- This is just a division! Simply compute x as:

$$x \equiv \frac{b}{a} \pmod{N}$$

(computation done using the extended Euclidean algorithm)
- In $\mathbb{Z}/N\mathbb{Z}$, DLP can be solved in polynomial time (in $\log N$), even though generic algorithms are all exponential

22/39
©2018 NTT Secure Platform Laboratories

Additive group DLP

- There are of course some groups when the DLP can be solved much faster than by generic techniques
- Example: the cyclic group $\mathbb{Z}/N\mathbb{Z}$ of integers modulo N under addition
- This is an additive group. The DLP is to find, given two elements $a, b \in \mathbb{Z}/N\mathbb{Z}$, a value x such that $b \equiv ax \pmod{N}$
- This is just a division! Simply compute x as:

$$x \equiv \frac{b}{a} \pmod{N}$$

(computation done using the extended Euclidean algorithm)
- In $\mathbb{Z}/N\mathbb{Z}$, DLP can be solved in polynomial time (in $\log N$), even though generic algorithms are all exponential

22/39
©2018 NTT Secure Platform Laboratories

Additive group DLP

- There are of course some groups when the DLP can be solved much faster than by generic techniques
- Example: the cyclic group $\mathbb{Z}/N\mathbb{Z}$ of integers modulo N under addition
- This is an additive group. The DLP is to find, given two elements $a, b \in \mathbb{Z}/N\mathbb{Z}$, a value x such that $b \equiv ax \pmod{N}$
- This is just a division! Simply compute x as:

$$x \equiv \frac{b}{a} \pmod{N}$$

(computation done using the extended Euclidean algorithm)
- In $\mathbb{Z}/N\mathbb{Z}$, DLP can be solved in polynomial time (in $\log N$), even though generic algorithms are all exponential

22/39
©2018 NTT Secure Platform Laboratories

Additive group DLP

- There are of course some groups when the DLP can be solved much faster than by generic techniques
- Example: the cyclic group $\mathbb{Z}/N\mathbb{Z}$ of integers modulo N under addition
- This is an additive group. The DLP is to find, given two elements $a, b \in \mathbb{Z}/N\mathbb{Z}$, a value x such that $b \equiv ax \pmod{N}$
- This is just a division! Simply compute x as:

$$x \equiv \frac{b}{a} \pmod{N}$$

  (computation done using the extended Euclidean algorithm)
- In $\mathbb{Z}/N\mathbb{Z}$, DLP can be solved in polynomial time (in $\log N$), even though generic algorithms are all exponential

Multiplicative group DLP

- Fix p a prime. The multiplicative group $G = (\mathbb{Z}/p\mathbb{Z})^\times$ is cyclic of order $p-1$ with some generator g
- We do not know how to solve the DLP in G in polynomial time, but we can still do much better than generic algorithms. The best known algorithms are subexponential
- More precisely, if p is n-bit long, the best algorithm (GNFS) has a complexity of $2^{\widetilde{O}(n^{1/3})}$, which is considerably less than Pollard's $\sqrt{p-1} \approx 2^{n/2}$
- In the next slide, brief description of a simpler subexponential algorithm, index calculus, based on similar principles. Complexity of $2^{\widetilde{O}(n^{1/2})}$

Multiplicative group DLP

- Fix p a prime. The multiplicative group $G = (\mathbb{Z}/p\mathbb{Z})^\times$ is cyclic of order $p-1$ with some generator g
- We do not know how to solve the DLP in G in polynomial time, but we can still do much better than generic algorithms. The best known algorithms are subexponential
- More precisely, if p is n-bit long, the best algorithm (GNFS) has a complexity of $2^{\widetilde{O}(n^{1/3})}$, which is considerably less than Pollard's $\sqrt{p-1} \approx 2^{n/2}$
- In the next slide, brief description of a simpler subexponential algorithm, index calculus, based on similar principles. Complexity of $2^{\widetilde{O}(n^{1/2})}$

23/39
©2018 NTT Secure Platform Laboratories

Multiplicative group DLP

- Fix p a prime. The multiplicative group $G = (\mathbb{Z}/p\mathbb{Z})^\times$ is cyclic of order $p-1$ with some generator g
- We do not know how to solve the DLP in G in polynomial time, but we can still do much better than generic algorithms. The best known algorithms are subexponential
- More precisely, if p is n-bit long, the best algorithm (GNFS) has a complexity of $2^{\widetilde{O}(n^{1/3})}$, which is considerably less than Pollard's $\sqrt{p-1} \approx 2^{n/2}$
- In the next slide, brief description of a simpler subexponential algorithm, index calculus, based on similar principles. Complexity of $2^{\widetilde{O}(n^{1/2})}$

23/39
©2018 NTT Secure Platform Laboratories

Multiplicative group DLP

- Fix p a prime. The multiplicative group $G = (\mathbb{Z}/p\mathbb{Z})^\times$ is cyclic of order $p-1$ with some generator g
- We do not know how to solve the DLP in G in polynomial time, but we can still do much better than generic algorithms. The best known algorithms are subexponential
- More precisely, if p is n-bit long, the best algorithm (GNFS) has a complexity of $2^{\widetilde{O}(n^{1/3})}$, which is considerably less than Pollard's $\sqrt{p-1} \approx 2^{n/2}$
- In the next slide, brief description of a simpler subexponential algorithm, index calculus, based on similar principles. Complexity of $2^{\widetilde{O}(n^{1/2})}$

Index calculus in $(\mathbb{Z}/p\mathbb{Z})^\times$

- Start by collecting all the small prime numbers $\ell_1 = 2, \ell_2 = 3, \cdots, \ell_r$ up to some bound B in a list F called the factor base. One can easily check if an integer has all its prime factors in F (and then factor it), and estimate the probability that this happens. Such a number is called B-smooth
- Now we will try to find the discrete logs $x_1, \ldots, x_r$ of all the elements of F wrt g. To do so, pick random numbers k_j and check if $g^{k_j} \bmod p$ is B-smooth. If so, we can factor it and get a relation:

$$g^{k_j} \equiv \ell_1^{e_{j,1}} \cdots \ell_r^{e_{j,r}} \pmod{p}$$

or after taking discrete logs:

$$k_j \equiv e_{j,1}x_1 + \cdots + e_{j,r}x_r \pmod{p-1}$$

If we find more than r such relations, applying Gaussian elimination allows to find $x_1, \ldots, x_r$

Index calculus in $(\mathbb{Z}/p\mathbb{Z})^\times$

- Start by collecting all the small prime numbers $\ell_1 = 2, \ell_2 = 3, \cdots, \ell_r$ up to some bound B in a list F called the factor base. One can easily check if an integer has all its prime factors in F (and then factor it), and estimate the probability that this happens. Such a number is called B-smooth
- Now we will try to find the discrete logs $x_1, \ldots, x_r$ of all the elements of F wrt g. To do so, pick random numbers k_j and check if $g^{k_j} \bmod p$ is B-smooth. If so, we can factor it and get a relation:

$$g^{k_j} \equiv \ell_1^{e_{j,1}} \cdots \ell_r^{e_{j,r}} \pmod{p}$$

or after taking discrete logs:

$$k_j \equiv e_{j,1}x_1 + \cdots + e_{j,r}x_r \pmod{p-1}$$

If we find more than r such relations, applying Gaussian elimination allows to find $x_1, \ldots, x_r$

Index calculus in $(\mathbb{Z}/p\mathbb{Z})^\times$

- After that, finding the discrete log of any $h \in G$ is comparatively easy. Just find a single random s such that $g^s \cdot h \bmod p$ is B-smooth, and factor it to get:

$$\begin{aligned} h &\equiv g^{-s} \cdot \ell_1^{e_1} \cdots \ell_r^{e_r} \\ &\equiv g^{-s+e_1x_1+\cdots+e_rx_r} \pmod{p} \end{aligned}$$

- For a well-chosen B, smoothness probability estimates allow to show that the overall complexity is $2^{\widetilde{O}(n^{1/2})}$

Index calculus in $(\mathbb{Z}/p\mathbb{Z})^\times$

- After that, finding the discrete log of any $h \in G$ is comparatively easy. Just find a single random s such that $g^s \cdot h \bmod p$ is B-smooth, and factor it to get:

$$\begin{aligned} h &\equiv g^{-s} \cdot \ell_1^{e_1} \cdots \ell_r^{e_r} \\ &\equiv g^{-s+e_1 x_1 + \cdots + e_r x_r} \pmod{p} \end{aligned}$$

- For a well-chosen B, smoothness probability estimates allow to show that the overall complexity is $2^{\widetilde{O}(n^{1/2})}$

DLP in finite fields

- The group $(\mathbb{Z}/p\mathbb{Z})^\times$ is the multiplicative group of the finite field $\mathbb{F}_p$
- One can also consider the DLP in the multiplicative group of larger finite fields $\mathbb{F}_q$, $q = p^m$. The GNFS algorithm extends to that setting, and always gives algorithms in $2^{\widetilde{O}(n^{1/3})}$ where n is the bit size of q
- Recent breakthrough [BGJT14]: if p is very small (e.g. constant), a refinement of the algorithm gives quasipolynomial complexity
  - one of the most important advances in number-theoretic algorithms in the past decade
  - with consequences on some flavors of elliptic curve crypto

DLP in finite fields

- The group $(\mathbb{Z}/p\mathbb{Z})^\times$ is the multiplicative group of the finite field $\mathbb{F}_p$
- One can also consider the DLP in the multiplicative group of larger finite fields $\mathbb{F}_q$, $q = p^m$. The GNFS algorithm extends to that setting, and always gives algorithms in $2^{\widetilde{O}(n^{1/3})}$ where n is the bit size of q
- Recent breakthrough [BGJT14]: if p is very small (e.g. constant), a refinement of the algorithm gives quasipolynomial complexity
  - one of the most important advances in number-theoretic algorithms in the past decade
  - with consequences on some flavors of elliptic curve crypto

26/39
©2018 NTT Secure Platform Laboratories

DLP in finite fields

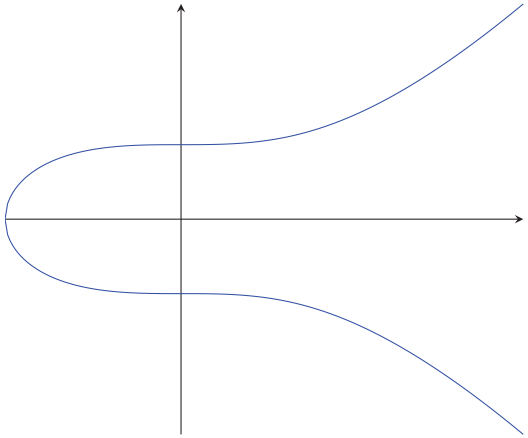
- The group $(\mathbb{Z}/p\mathbb{Z})^\times$ is the multiplicative group of the finite field $\mathbb{F}_p$
- One can also consider the DLP in the multiplicative group of larger finite fields $\mathbb{F}_q$, $q = p^m$. The GNFS algorithm extends to that setting, and always gives algorithms in $2^{\widetilde{O}(n^{1/3})}$ where n is the bit size of q
- Recent breakthrough [BGJT14]: if p is very small (e.g. constant), a refinement of the algorithm gives quasipolynomial complexity
  - one of the most important advances in number-theoretic algorithms in the past decade
  - with consequences on some flavors of elliptic curve crypto

26/39
©2018 NTT Secure Platform Laboratories

Outline

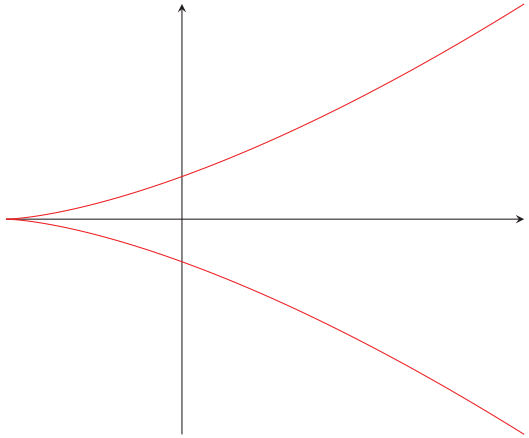
Outline

Elliptic curves



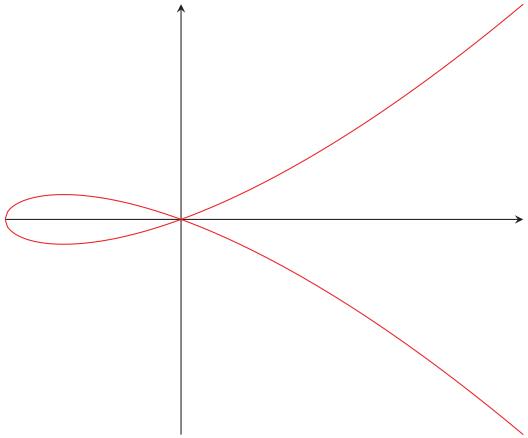
Elliptic curve: plane curve E of the form $y^2 = x^3 + ax + b$

Elliptic curves



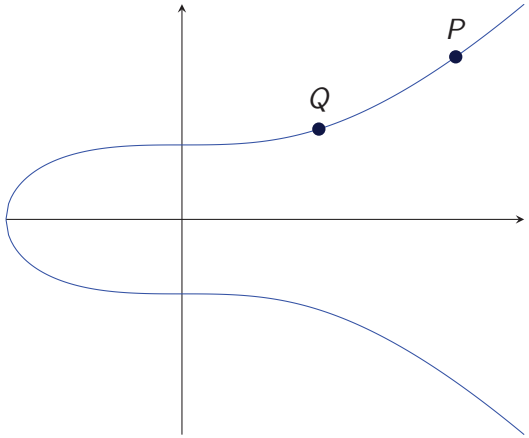
Must be non-singular, so not this

Elliptic curves



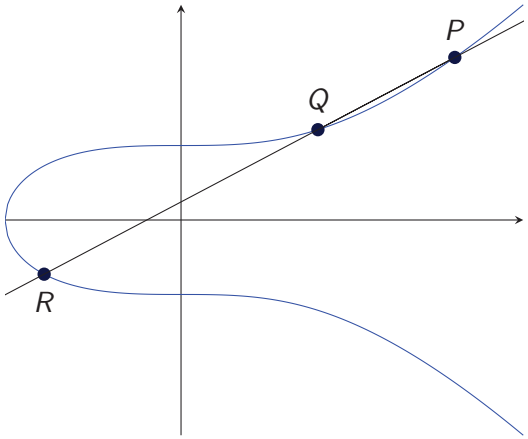
. . . nor this

Elliptic curves



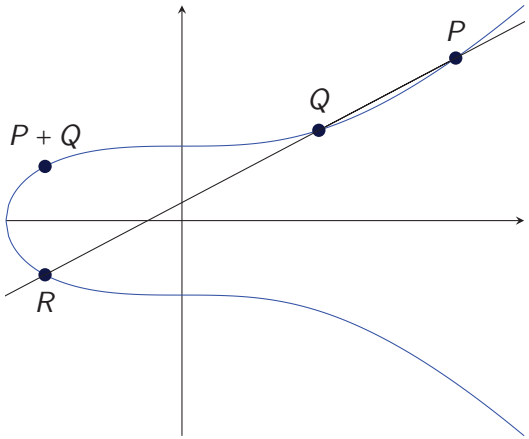
Take two points P, Q on E

Elliptic curves



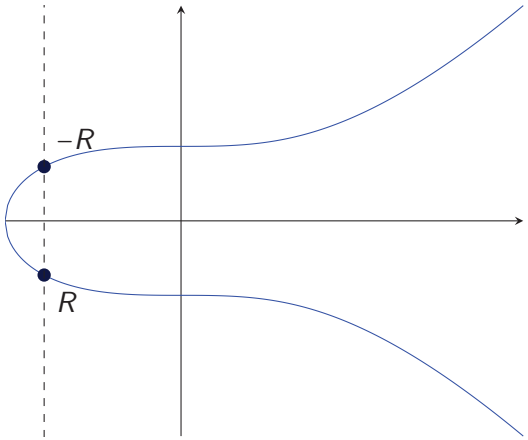
Because E is of degree 3, the line through P, Q intersects E at exactly one other point (with coordinates in the same field)

Elliptic curves



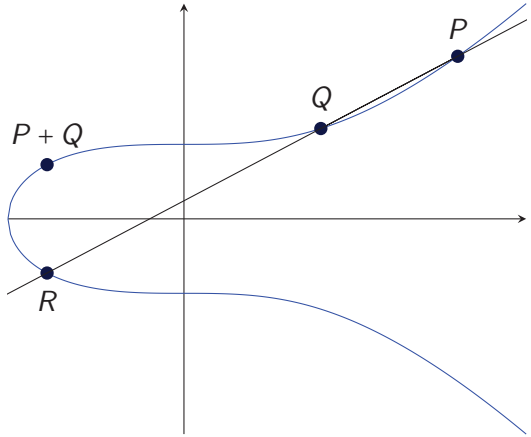
We define a group law on the points of E by defining $P + Q$ as the mirror image of R

Elliptic curves



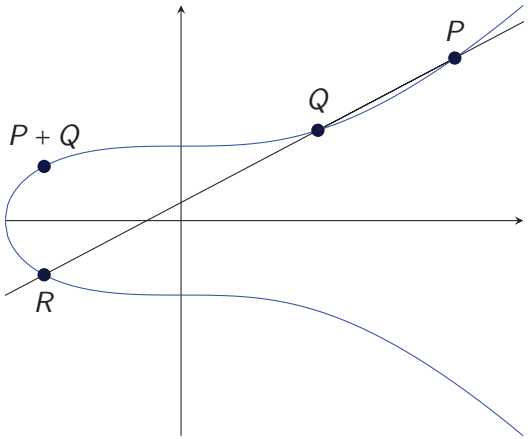
The identity element is at infinity along the y-axis; $R + (-R) = 0$

Elliptic curves



Commutative group: $P + Q = Q + P = -R$.
Associativity? Not so obvious but can be checked

Elliptic curves



These pictures are over the reals,
but everything works over finite fields too!

The ECDLP

- Consider $E : y^2 = x^3 + ax + b$ elliptic curve with coefficients a, b in the finite field $\mathbb{F}_q$
- As we saw, the set $E(\mathbb{F}_q)$ of points on E with coordinates in $\mathbb{F}_q$ (incl. the point at infinity) is a finite commutative group
  - its order is always of the form $q + O(\sqrt{q})$ (Hasse bound)
- Given a point $P \in E(\mathbb{F}_q)$ and an integer x, one can efficiently compute the scalar multiplication $[x]P = P + P + \cdots + P$ (usual group exponentiation)
- The elliptic curve discrete logarithm problem (ECDLP), on which all elliptic curve crypto is based, is the DLP in cyclic subgroups of $E(\mathbb{F}_q)$.
- Namely: given $P, Q \in E(\mathbb{F}_q)$ such that there exists x with $Q = [x]P$, find x

The ECDLP

- Consider $E : y^2 = x^3 + ax + b$ elliptic curve with coefficients a, b in the finite field $\mathbb{F}_q$
- As we saw, the set $E(\mathbb{F}_q)$ of points on E with coordinates in $\mathbb{F}_q$ (incl. the point at infinity) is a finite commutative group
  - its order is always of the form $q + O(\sqrt{q})$ (Hasse bound)
- Given a point $P \in E(\mathbb{F}_q)$ and an integer x, one can efficiently compute the scalar multiplication $[x]P = P + P + \cdots + P$ (usual group exponentiation)
- The elliptic curve discrete logarithm problem (ECDLP), on which all elliptic curve crypto is based, is the DLP in cyclic subgroups of $E(\mathbb{F}_q)$.
- Namely: given $P, Q \in E(\mathbb{F}_q)$ such that there exists x with $Q = [x]P$, find x

The ECDLP

- Consider $E : y^2 = x^3 + ax + b$ elliptic curve with coefficients a, b in the finite field $\mathbb{F}_q$
- As we saw, the set $E(\mathbb{F}_q)$ of points on E with coordinates in $\mathbb{F}_q$ (incl. the point at infinity) is a finite commutative group
  - its order is always of the form $q + O(\sqrt{q})$ (Hasse bound)
- Given a point $P \in E(\mathbb{F}_q)$ and an integer x, one can efficiently compute the scalar multiplication $[x]P = P + P + \cdots + P$ (usual group exponentiation)
- The elliptic curve discrete logarithm problem (ECDLP), on which all elliptic curve crypto is based, is the DLP in cyclic subgroups of $E(\mathbb{F}_q)$.
- Namely: given $P, Q \in E(\mathbb{F}_q)$ such that there exists x with $Q = [x]P$, find x

30/39
©2018 NTT Secure Platform Laboratories

The ECDLP

- Consider $E : y^2 = x^3 + ax + b$ elliptic curve with coefficients a, b in the finite field $\mathbb{F}_q$
- As we saw, the set $E(\mathbb{F}_q)$ of points on E with coordinates in $\mathbb{F}_q$ (incl. the point at infinity) is a finite commutative group
  - its order is always of the form $q + O(\sqrt{q})$ (Hasse bound)
- Given a point $P \in E(\mathbb{F}_q)$ and an integer x, one can efficiently compute the scalar multiplication $[x]P = P + P + \cdots + P$ (usual group exponentiation)
- The elliptic curve discrete logarithm problem (ECDLP), on which all elliptic curve crypto is based, is the DLP in cyclic subgroups of $E(\mathbb{F}_q)$.
- Namely: given $P, Q \in E(\mathbb{F}_q)$ such that there exists x with $Q = [x]P$, find x

30/39
©2018 NTT Secure Platform Laboratories

The ECDLP

- Consider $E : y^2 = x^3 + ax + b$ elliptic curve with coefficients a, b in the finite field $\mathbb{F}_q$
- As we saw, the set $E(\mathbb{F}_q)$ of points on E with coordinates in $\mathbb{F}_q$ (incl. the point at infinity) is a finite commutative group
  - its order is always of the form $q + O(\sqrt{q})$ (Hasse bound)
- Given a point $P \in E(\mathbb{F}_q)$ and an integer x, one can efficiently compute the scalar multiplication $[x]P = P + P + \cdots + P$ (usual group exponentiation)
- The elliptic curve discrete logarithm problem (ECDLP), on which all elliptic curve crypto is based, is the DLP in cyclic subgroups of $E(\mathbb{F}_q)$.
- Namely: given $P, Q \in E(\mathbb{F}_q)$ such that there exists x with $Q = [x]P$, find x

30/39
©2018 NTT Secure Platform Laboratories

Security of ECDLP

- For most elliptic curves, we know no better (classical) attacks on the ECDLP than the generic ones
- In particular, the complexity should be $O(\sqrt{\ell})$ where ℓ is the largest prime factor of the order $\#E(\mathbb{F}_q)$. If we choose $\#E(\mathbb{F}_q)$ as a prime or almost a prime, this is simply $O(\sqrt{q})$
- In other words, to get e.g. 128 bits of security, simply use an elliptic curve over a field of ≈ 256 bits
- In contrast, due to the subexponential attacks, to get the same level of security for the DLP in $(\mathbb{Z}/p\mathbb{Z})^\times$, one needs p of ≈ 3000 bits
- This is why elliptic curves are generally much more efficient
- However, weak elliptic curves do exist!

Security of ECDLP

- For most elliptic curves, we know no better (classical) attacks on the ECDLP than the generic ones
- In particular, the complexity should be $O(\sqrt{\ell})$ where ℓ is the largest prime factor of the order $\#E(\mathbb{F}_q)$. If we choose $\#E(\mathbb{F}_q)$ as a prime or almost a prime, this is simply $O(\sqrt{q})$
- In other words, to get e.g. 128 bits of security, simply use an elliptic curve over a field of ≈ 256 bits
- In contrast, due to the subexponential attacks, to get the same level of security for the DLP in $(\mathbb{Z}/p\mathbb{Z})^\times$, one needs p of ≈ 3000 bits
- This is why elliptic curves are generally much more efficient
- However, weak elliptic curves do exist!

31/39
©2018 NTT Secure Platform Laboratories

Security of ECDLP

- For most elliptic curves, we know no better (classical) attacks on the ECDLP than the generic ones
- In particular, the complexity should be $O(\sqrt{\ell})$ where ℓ is the largest prime factor of the order $\#E(\mathbb{F}_q)$. If we choose $\#E(\mathbb{F}_q)$ as a prime or almost a prime, this is simply $O(\sqrt{q})$
- In other words, to get e.g. 128 bits of security, simply use an elliptic curve over a field of ≈ 256 bits
- In contrast, due to the subexponential attacks, to get the same level of security for the DLP in $(\mathbb{Z}/p\mathbb{Z})^\times$, one needs p of ≈ 3000 bits
- This is why elliptic curves are generally much more efficient
- However, weak elliptic curves do exist!

31/39
©2018 NTT Secure Platform Laboratories

Security of ECDLP

- For most elliptic curves, we know no better (classical) attacks on the ECDLP than the generic ones
- In particular, the complexity should be $O(\sqrt{\ell})$ where ℓ is the largest prime factor of the order $\#E(\mathbb{F}_q)$. If we choose $\#E(\mathbb{F}_q)$ as a prime or almost a prime, this is simply $O(\sqrt{q})$
- In other words, to get e.g. 128 bits of security, simply use an elliptic curve over a field of ≈ 256 bits
- In contrast, due to the subexponential attacks, to get the same level of security for the DLP in $(\mathbb{Z}/p\mathbb{Z})^\times$, one needs p of ≈ 3000 bits
- This is why elliptic curves are generally much more efficient
- However, weak elliptic curves do exist!

31/39
©2018 NTT Secure Platform Laboratories

Security of ECDLP

- For most elliptic curves, we know no better (classical) attacks on the ECDLP than the generic ones
- In particular, the complexity should be $O(\sqrt{\ell})$ where ℓ is the largest prime factor of the order $\#E(\mathbb{F}_q)$. If we choose $\#E(\mathbb{F}_q)$ as a prime or almost a prime, this is simply $O(\sqrt{q})$
- In other words, to get e.g. 128 bits of security, simply use an elliptic curve over a field of ≈ 256 bits
- In contrast, due to the subexponential attacks, to get the same level of security for the DLP in $(\mathbb{Z}/p\mathbb{Z})^\times$, one needs p of ≈ 3000 bits
- This is why elliptic curves are generally much more efficient
- However, weak elliptic curves do exist!

Security of ECDLP

- For most elliptic curves, we know no better (classical) attacks on the ECDLP than the generic ones
- In particular, the complexity should be $O(\sqrt{\ell})$ where ℓ is the largest prime factor of the order $\#E(\mathbb{F}_q)$. If we choose $\#E(\mathbb{F}_q)$ as a prime or almost a prime, this is simply $O(\sqrt{q})$
- In other words, to get e.g. 128 bits of security, simply use an elliptic curve over a field of ≈ 256 bits
- In contrast, due to the subexponential attacks, to get the same level of security for the DLP in $(\mathbb{Z}/p\mathbb{Z})^\times$, one needs p of ≈ 3000 bits
- This is why elliptic curves are generally much more efficient
- However, weak elliptic curves do exist!

Transporting discrete logs under homomorphism

- Recall that a homomorphism φ between two groups G and H is a mapping $G \to H$ such that for $g_1, g_2 \in G$, $\varphi(g_1 g_2) = \varphi(g_1)\varphi(g_2)$
- Most of the non-generic attacks on the ECDLP take the following form:
  1. Suppose we want to solve the ECDLP for points $P, Q \in E(\mathbb{F}_q)$ of prime order ℓ
  2. Construct an efficiently computable homomorphism $\varphi : \langle P \rangle \to G$ such that the DLP in G is "easy" (and $\varphi(P) \neq 1$)
  3. Then, the relation $Q = [x]P$ gives $\varphi(Q) = \varphi(P)^x$ and solving the DLP in G reveals x
- Of course, this can only be done for curves E verifying some special properties
- The condition that φ is efficiently computable is essential
  - indeed, the discrete logarithm function $\langle P \rangle \to \mathbb{Z}/\ell\mathbb{Z}$ itself is a homomorphism!

Transporting discrete logs under homomorphism

- Recall that a homomorphism φ between two groups G and H is a mapping $G \to H$ such that for $g_1, g_2 \in G$, $\varphi(g_1 g_2) = \varphi(g_1)\varphi(g_2)$
- Most of the non-generic attacks on the ECDLP take the following form:
  1. Suppose we want to solve the ECDLP for points $P, Q \in E(\mathbb{F}_q)$ of prime order ℓ
  2. Construct an efficiently computable homomorphism $\varphi : \langle P \rangle \to G$ such that the DLP in G is "easy" (and $\varphi(P) \neq 1$)
  3. Then, the relation $Q = [x]P$ gives $\varphi(Q) = \varphi(P)^x$ and solving the DLP in G reveals x
- Of course, this can only be done for curves E verifying some special properties
- The condition that φ is efficiently computable is essential
  - indeed, the discrete logarithm function $\langle P \rangle \to \mathbb{Z}/\ell\mathbb{Z}$ itself is a homomorphism!

Transporting discrete logs under homomorphism

- Recall that a homomorphism φ between two groups G and H is a mapping $G \to H$ such that for $g_1, g_2 \in G$, $\varphi(g_1 g_2) = \varphi(g_1)\varphi(g_2)$
- Most of the non-generic attacks on the ECDLP take the following form:
  1. Suppose we want to solve the ECDLP for points $P, Q \in E(\mathbb{F}_q)$ of prime order ℓ
  2. Construct an efficiently computable homomorphism $\varphi\colon \langle P \rangle \to G$ such that the DLP in G is "easy" (and $\varphi(P) \neq 1$)
  3. Then, the relation $Q = [x]P$ gives $\varphi(Q) = \varphi(P)^x$ and solving the DLP in G reveals x
- Of course, this can only be done for curves E verifying some special properties
- The condition that φ is efficiently computable is essential
  - indeed, the discrete logarithm function $\langle P \rangle \to \mathbb{Z}/\ell\mathbb{Z}$ itself is a homomorphism!

Transporting discrete logs under homomorphism

- Recall that a homomorphism φ between two groups G and H is a mapping $G \to H$ such that for $g_1, g_2 \in G$, $\varphi(g_1 g_2) = \varphi(g_1)\varphi(g_2)$
- Most of the non-generic attacks on the ECDLP take the following form:
  1. Suppose we want to solve the ECDLP for points $P, Q \in E(\mathbb{F}_q)$ of prime order ℓ
  2. Construct an efficiently computable homomorphism $\varphi\colon \langle P \rangle \to G$ such that the DLP in G is "easy" (and $\varphi(P) \neq 1$)
  3. Then, the relation $Q = [x]P$ gives $\varphi(Q) = \varphi(P)^x$ and solving the DLP in G reveals x
- Of course, this can only be done for curves E verifying some special properties
- The condition that φ is efficiently computable is essential
  - indeed, the discrete logarithm function $\langle P \rangle \to \mathbb{Z}/\ell\mathbb{Z}$ itself is a homomorphism!

32/39
©2018 NTT Secure Platform Laboratories

Transporting discrete logs under homomorphism

- Recall that a homomorphism φ between two groups G and H is a mapping $G \to H$ such that for $g_1, g_2 \in G$, $\varphi(g_1 g_2) = \varphi(g_1)\varphi(g_2)$
- Most of the non-generic attacks on the ECDLP take the following form:
  1. Suppose we want to solve the ECDLP for points $P, Q \in E(\mathbb{F}_q)$ of prime order ℓ
  2. Construct an efficiently computable homomorphism $\varphi\colon \langle P \rangle \to G$ such that the DLP in G is "easy" (and $\varphi(P) \neq 1$)
  3. Then, the relation $Q = [x]P$ gives $\varphi(Q) = \varphi(P)^x$ and solving the DLP in G reveals x
- Of course, this can only be done for curves E verifying some special properties
- The condition that φ is efficiently computable is essential
  - indeed, the discrete logarithm function $\langle P \rangle \to \mathbb{Z}/\ell\mathbb{Z}$ itself is a homomorphism!

32/39
©2018 NTT Secure Platform Laboratories

Transporting discrete logs under homomorphism

- Recall that a homomorphism φ between two groups G and H is a mapping $G \to H$ such that for $g_1, g_2 \in G$, $\varphi(g_1 g_2) = \varphi(g_1)\varphi(g_2)$
- Most of the non-generic attacks on the ECDLP take the following form:
  1. Suppose we want to solve the ECDLP for points $P, Q \in E(\mathbb{F}_q)$ of prime order ℓ
  2. Construct an efficiently computable homomorphism $\varphi\colon \langle P \rangle \to G$ such that the DLP in G is "easy" (and $\varphi(P) \neq 1$)
  3. Then, the relation $Q = [x]P$ gives $\varphi(Q) = \varphi(P)^x$ and solving the DLP in G reveals x
- Of course, this can only be done for curves E verifying some special properties
- The condition that φ is efficiently computable is essential
  - indeed, the discrete logarithm function $\langle P \rangle \to \mathbb{Z}/\ell\mathbb{Z}$ itself is a homomorphism!

32/39
©2018 NTT Secure Platform Laboratories

Transporting discrete logs under homomorphism

- Recall that a homomorphism φ between two groups G and H is a mapping $G \to H$ such that for $g_1, g_2 \in G$, $\varphi(g_1 g_2) = \varphi(g_1)\varphi(g_2)$
- Most of the non-generic attacks on the ECDLP take the following form:
  1. Suppose we want to solve the ECDLP for points $P, Q \in E(\mathbb{F}_q)$ of prime order ℓ
  2. Construct an efficiently computable homomorphism $\varphi\colon \langle P \rangle \to G$ such that the DLP in G is "easy" (and $\varphi(P) \neq 1$)
  3. Then, the relation $Q = [x]P$ gives $\varphi(Q) = \varphi(P)^x$ and solving the DLP in G reveals x
- Of course, this can only be done for curves E verifying some special properties
- The condition that φ is efficiently computable is essential
  - indeed, the discrete logarithm function $\langle P \rangle \to \mathbb{Z}/\ell\mathbb{Z}$ itself is a homomorphism!

Outline

The Menezes–Okamoto–Vanstone attack

- The epitome of attacks on ECDLP by transporting discrete logs to a weaker group is the MOV attack, relying on the existence of pairings on elliptic curves
- Weil: E elliptic curve over $\mathbb{F}_q$, ℓ prime not dividing q. There exists a non-degenerate bilinear pairing:

$$e\colon E[\ell] \times E[\ell] \to \mu_\ell$$

  where $E[\ell]$ is the group of points on E of order dividing ℓ, and μ_ℓ is the group of ℓ-th roots of unity (both possibly in some extension of $\mathbb{F}_q$)
  - bilinearity: $e(P + P', Q) = e(P, Q) \cdot e(P', Q)$ and similarly on the right
  - non-degeneracy: for any P of exact order ℓ, there exists Q such that $e(P, Q) \neq 1$
- Miller: we can compute $e(P, Q)$ in $\mathrm{polylog}(\ell)$ operations in the fields of definition of P, Q and μ_ℓ

The Menezes–Okamoto–Vanstone attack

- The epitome of attacks on ECDLP by transporting discrete logs to a weaker group is the MOV attack, relying on the existence of pairings on elliptic curves
- Weil: E elliptic curve over $\mathbb{F}_q$, ℓ prime not dividing q. There exists a non-degenerate bilinear pairing:

$$e\colon E[\ell] \times E[\ell] \to \mu_\ell$$

  where $E[\ell]$ is the group of points on E of order dividing ℓ, and μ_ℓ is the group of ℓ-th roots of unity (both possibly in some extension of $\mathbb{F}_q$)
  - bilinearity: $e(P + P', Q) = e(P, Q) \cdot e(P', Q)$ and similarly on the right
  - non-degeneracy: for any P of exact order ℓ, there exists Q such that $e(P, Q) \neq 1$
- Miller: we can compute $e(P, Q)$ in $\mathrm{polylog}(\ell)$ operations in the fields of definition of P, Q and μ_ℓ

The Menezes–Okamoto–Vanstone attack

- The epitome of attacks on ECDLP by transporting discrete logs to a weaker group is the MOV attack, relying on the existence of pairings on elliptic curves
- Weil: E elliptic curve over $\mathbb{F}_q$, ℓ prime not dividing q. There exists a non-degenerate bilinear pairing:

$$e\colon E[\ell] \times E[\ell] \to \mu_\ell$$

  where $E[\ell]$ is the group of points on E of order dividing ℓ, and μ_ℓ is the group of ℓ-th roots of unity (both possibly in some extension of $\mathbb{F}_q$)
  - bilinearity: $e(P + P', Q) = e(P, Q) \cdot e(P', Q)$ and similarly on the right
  - non-degeneracy: for any P of exact order ℓ, there exists Q such that $e(P, Q) \neq 1$
- Miller: we can compute $e(P, Q)$ in $\mathrm{polylog}(\ell)$ operations in the fields of definition of P, Q and μ_ℓ

The Menezes–Okamoto–Vanstone attack (desc.)

- Given any elliptic curve E and any P of order ℓ on E, we therefore get an efficient(?) homomorphism $\langle P \rangle \to \mu_\ell$ as follows:
  1. Find $Q \in E[\ell]$ satisfying non-degeneracy wrt P;
  2. Then $R \mapsto e(R, Q)$ is a homomorphism $\langle P \rangle \to \mu_\ell$, "efficient" by Miller's algorithm

  Since μ_ℓ is a subgroup of the multiplicative group of a finite field, and those multiplicative groups have subexponential DLP, this gives a subexponential algorithm for the ECDLP?!
- Of course, there is a catch! The field of definition of μ_ℓ and Q are usually very large—exponentially so in fact!
  - $\mu_\ell \subset \mathbb{F}_{q^k}^*$ implies that ℓ divides $q^k - 1$, and for a "random" ℓ, the smallest such k is roughly as large as ℓ itself!

The Menezes–Okamoto–Vanstone attack (desc.)

- Given any elliptic curve E and any P of order ℓ on E, we therefore get an efficient(?) homomorphism $\langle P \rangle \to \mu_\ell$ as follows:
  1. Find $Q \in E[\ell]$ satisfying non-degeneracy wrt P;
  2. Then $R \mapsto e(R, Q)$ is a homomorphism $\langle P \rangle \to \mu_\ell$, "efficient" by Miller's algorithm

  Since μ_ℓ is a subgroup of the multiplicative group of a finite field, and those multiplicative groups have subexponential DLP, this gives a subexponential algorithm for the ECDLP?!
- Of course, there is a catch! The field of definition of μ_ℓ and Q are usually very large—exponentially so in fact!
  - $\mu_\ell \subset \mathbb{F}_{q^k}^*$ implies that ℓ divides $q^k - 1$, and for a "random" ℓ, the smallest such k is roughly as large as ℓ itself!

The Menezes–Okamoto–Vanstone attack (desc.)

- Given any elliptic curve E and any P of order ℓ on E, we therefore get an efficient(?) homomorphism $\langle P \rangle \to \mu_\ell$ as follows:
  1. Find $Q \in E[\ell]$ satisfying non-degeneracy wrt P;
  2. Then $R \mapsto e(R, Q)$ is a homomorphism $\langle P \rangle \to \mu_\ell$, "efficient" by Miller's algorithm

  Since μ_ℓ is a subgroup of the multiplicative group of a finite field, and those multiplicative groups have subexponential DLP, this gives a subexponential algorithm for the ECDLP?!
- Of course, there is a catch! The field of definition of μ_ℓ and Q are usually very large—exponentially so in fact!
  - $\mu_\ell \subset \mathbb{F}_{q^k}^*$ implies that ℓ divides $q^k - 1$, and for a "random" ℓ, the smallest such k is roughly as large as ℓ itself!

The Menezes–Okamoto–Vanstone attack (desc.)

- Given any elliptic curve E and any P of order ℓ on E, we therefore get an efficient(?) homomorphism $\langle P\rangle \to \mu_\ell$ as follows:
  1. Find $Q \in E[\ell]$ satisfying non-degeneracy wrt P;
  2. Then $R \mapsto e(R,Q)$ is a homomorphism $\langle P\rangle \to \mu_\ell$, "efficient" by Miller's algorithm

  Since μ_ℓ is a subgroup of the multiplicative group of a finite field, and those multiplicative groups have subexponential DLP, this gives a subexponential algorithm for the ECDLP?!
- Of course, there is a catch! The field of definition of μ_ℓ and Q are usually very large—exponentially so in fact!
  - $\mu_\ell \subset \mathbb{F}_{q^k}^*$ implies that ℓ divides $q^k - 1$, and for a "random" ℓ, the smallest such k is roughly as large as ℓ itself!

Practical cases of MOV

- However, in some cases, the MOV attack is actually efficient
- Example: $E : y^2 = x^3 + ax$ over a field $\mathbb{F}_p$ with $p \equiv 3 \mod 4$. It is an easy exercise (using the fact that -1 is not a square in $\mathbb{F}_p$ and that the polynomial $x^3 - ax$ is odd) to check that $\#E(\mathbb{F}_p) = p + 1$
- Thus, for any $\ell | \#E(\mathbb{F}_p)$, we have $\ell | p^2 - 1$. Hence, the entire Weil pairing is defined over the field $\mathbb{F}_{p^2}$: easy computations
- This allows to transfer the ECDLP on E to the discrete logarithm in $\mathbb{F}_{p^2}^*$, which can be solved in subexponential time using GNFS.
- That curve E is dangerous...

Practical cases of MOV

- However, in some cases, the MOV attack is actually efficient
- Example: $E : y^2 = x^3 + ax$ over a field $\mathbb{F}_p$ with $p \equiv 3 \mod 4$. It is an easy exercise (using the fact that -1 is not a square in $\mathbb{F}_p$ and that the polynomial $x^3 - ax$ is odd) to check that $\#E(\mathbb{F}_p) = p + 1$
- Thus, for any $\ell | \#E(\mathbb{F}_p)$, we have $\ell | p^2 - 1$. Hence, the entire Weil pairing is defined over the field $\mathbb{F}_{p^2}$: easy computations
- This allows to transfer the ECDLP on E to the discrete logarithm in $\mathbb{F}_{p^2}^*$, which can be solved in subexponential time using GNFS.
- That curve E is dangerous...

Practical cases of MOV

- However, in some cases, the MOV attack is actually efficient
- Example: $E : y^2 = x^3 + ax$ over a field $\mathbb{F}_p$ with $p \equiv 3 \mod 4$. It is an easy exercise (using the fact that -1 is not a square in $\mathbb{F}_p$ and that the polynomial $x^3 - ax$ is odd) to check that $\#E(\mathbb{F}_p) = p + 1$
- Thus, for any $\ell | \#E(\mathbb{F}_p)$, we have $\ell | p^2 - 1$. Hence, the entire Weil pairing is defined over the field $\mathbb{F}_{p^2}$: easy computations
- This allows to transfer the ECDLP on E to the discrete logarithm in $\mathbb{F}_{p^2}^*$, which can be solved in subexponential time using GNFS.
- That curve E is dangerous...

Practical cases of MOV

- However, in some cases, the MOV attack is actually efficient
- Example: $E : y^2 = x^3 + ax$ over a field $\mathbb{F}_p$ with $p \equiv 3 \mod 4$. It is an easy exercise (using the fact that -1 is not a square in $\mathbb{F}_p$ and that the polynomial $x^3 - ax$ is odd) to check that $\#E(\mathbb{F}_p) = p + 1$
- Thus, for any $\ell | \#E(\mathbb{F}_p)$, we have $\ell | p^2 - 1$. Hence, the entire Weil pairing is defined over the field $\mathbb{F}_{p^2}$: easy computations
- This allows to transfer the ECDLP on E to the discrete logarithm in $\mathbb{F}_{p^2}^*$, which can be solved in subexponential time using GNFS.
- That curve E is dangerous...

Practical cases of MOV

- However, in some cases, the MOV attack is actually efficient
- Example: $E : y^2 = x^3 + ax$ over a field $\mathbb{F}_p$ with $p \equiv 3 \mod 4$. It is an easy exercise (using the fact that -1 is not a square in $\mathbb{F}_p$ and that the polynomial $x^3 - ax$ is odd) to check that $\#E(\mathbb{F}_p) = p + 1$
- Thus, for any $\ell | \#E(\mathbb{F}_p)$, we have $\ell | p^2 - 1$. Hence, the entire Weil pairing is defined over the field $\mathbb{F}_{p^2}$: easy computations
- This allows to transfer the ECDLP on E to the discrete logarithm in $\mathbb{F}_{p^2}^*$, which can be solved in subexponential time using GNFS.
- That curve E is dangerous...

36/39
©2018 NTT Secure Platform Laboratories

Pairing-friendly curve

- The only case where one would use such a curve E is when we specifically want to compute the pairing: pairing-based cryptography
- The curves used in this case are said to be pairing-friendly
- MOV is an unavoidable, but "useful", consequence of the existence of the pairing
- For other applications, easy to rule out: just verify that k such that $\ell | p^k - 1$ is exponentially large

Pairing-friendly curve

- The only case where one would use such a curve E is when we specifically want to compute the pairing: pairing-based cryptography
- The curves used in this case are said to be pairing-friendly
- MOV is an unavoidable, but "useful", consequence of the existence of the pairing
- For other applications, easy to rule out: just verify that k such that $\ell | p^k - 1$ is exponentially large

37/39
©2018 NTT Secure Platform Laboratories

Pairing-friendly curve

- The only case where one would use such a curve E is when we specifically want to compute the pairing: pairing-based cryptography
- The curves used in this case are said to be pairing-friendly
- MOV is an unavoidable, but "useful", consequence of the existence of the pairing
- For other applications, easy to rule out: just verify that k such that $\ell | p^k - 1$ is exponentially large

Pairing-friendly curve

- The only case where one would use such a curve E is when we specifically want to compute the pairing: pairing-based cryptography
- The curves used in this case are said to be pairing-friendly
- MOV is an unavoidable, but "useful", consequence of the existence of the pairing
- For other applications, easy to rule out: just verify that k such that $\ell | p^k - 1$ is exponentially large

Outline

Other weak elliptic curves

- Anomalous curves: $\#E(\mathbb{F}_p) = p$. Poly-time ECDLP via p-adic analysis
- GHS attack for curves over extension fields; not stable under isogenies
- Attempts to adapt index calculus-style attacks to the elliptic curve setting (Diem et al.). Give subexponential asymptotic complexity in some cases (but not practical)

Other weak elliptic curves

- Anomalous curves: $\#E(\mathbb{F}_p) = p$. Poly-time ECDLP via p-adic analysis
- GHS attack for curves over extension fields; not stable under isogenies
- Attempts to adapt index calculus-style attacks to the elliptic curve setting (Diem et al.). Give subexponential asymptotic complexity in some cases (but not practical)

Other weak elliptic curves

- Anomalous curves: $\#E(\mathbb{F}_p) = p$. Poly-time ECDLP via p-adic analysis
- GHS attack for curves over extension fields; not stable under isogenies
- Attempts to adapt index calculus-style attacks to the elliptic curve setting (Diem et al.). Give subexponential asymptotic complexity in some cases (but not practical)

39/39
©2018 NTT Secure Platform Laboratories