CS 6501 Week 5: Elliptic-Curve Cryptography

In previous weeks, we saw how to use the hardness of problems like discrete log, CDH, and DDH to construct public-key cryptography. In all of these cases, we require working over a suitable group where these assumptions plausibly hold. So far, we have mostly worked with a prime-order subgroup of $\mathbb{Z}_p^*$ (where $p = 2q + 1$).

But just how difficult is the discrete logarithm problem in $\mathbb{Z}_p^*$?

$\tilde{O}(\cdot)$ notation suppresses polylogarithmic terms

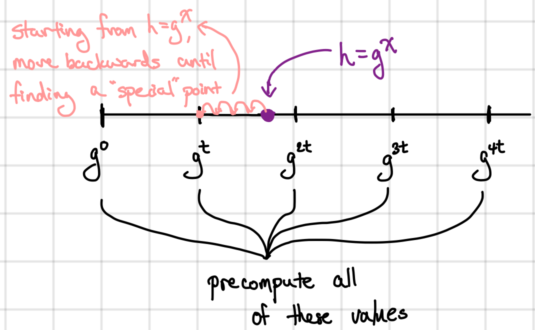
- Best known algorithm (based on general number field sieve (GNFS)) runs in time roughly $2^{\tilde{O}(\sqrt[3]{\log p})}$ — $\log p$ is the bitlength of the prime p
  - [$2^{\tilde{O}(\sqrt[3]{\log p})}$ is a <u>subexponential-time</u> algorithm] — running time bounded by $2^{(n^{\varepsilon})}$ for some $\varepsilon < 1$.
- Concretely, to get "128-bits" of security (e.g., security comparable to AES-128), recommendation is to use a <u>3092-bit</u> modulus $\rightarrow$ public-key operations become substantially slower than symmetric ones
  - [Using RSA is no better... Also need comparably-sized modulus for security]
    - In fact, the best algorithm for factoring / RSA is also the GNFS and <u>remarkably</u>, advancements in better factoring algorithms have proceeded almost in lockstep with advancements in discrete log algorithms!
- Is this the best we can do? If I give you an arbitrary group $\mathbb{G}$ of order p, how hard is discrete log in $\mathbb{G}$?
  - the GNFS is not a generic algorithm and works because the elements in $\mathbb{Z}_p^*$ are integers

For "generic" algorithms (i.e., algorithms that use the group operation as a black box), Shoup showed that the <u>best</u> discrete log algorithm runs in time $2^{\log p/2}$ – namely, <u>exponential</u> in the size of the group.

<u>Baby-Step/Giant-Step Algorithm</u>: Let $\mathbb{G}$ be a group of prime order p with generator g. Given a discrete log instance (g, h), the algorithm works as follows:

1. Let $t = \sqrt{p} = 2^{\frac{\log p}{2}}$
2. Compute the sequence $a_i = g^{(i \cdot t)}$ for $i = 0, \ldots, t-1$ ["giant" step]
3. For each $i = 0, \ldots, t^{-1}$, check if $h \cdot g^{-i} = a_j$ for some $j = 0, 1, \ldots, t-1$ ["baby" step]
   - If so output the discrete log <u>$x = jt + i$</u>

To see why this algorithm works, observe that if $h = {}^x$, then we can always write $x = jt + i$ for $0 \leq i, j < t$ where $t = \sqrt{p}$.



Observe: this algorithm only requires a way to evaluate the group operation and does not need information on how the group is represented (this is a "generic" algorithm)

Running time of this algorithm is $\tilde{O}(\sqrt{p})$, and space complexity is also $\tilde{O}(\sqrt{p})$. Using Floyd's cycle finding ("slow pointer"/"fast pointer") algorithm, we can obtain an algorithm with the same running time but $\tilde{O}(1)$ space (Pollard's rho algorithm).

- These generic algorithms for discrete log match Shoup's lower bound for discrete log.
- <u>Question</u>: Are there candidate groups where generic algorithms are the <u>best-known</u> algorithm? If so, we can potentially set the group size to be $p = 2\lambda = 256$ (to get 128-bits of security).

Elliptic-curve groups: a candidate group where the best known discrete log algorithms are the generic ones
↳ Studied by mathematicians since antiquity! [See work of Diophantus, circa 200 AD]
↳ Proposed for use in cryptographic applications in the 1980s ⟶ now is a leading choice for public-key cryptography on the web [another example where abstract concepts in mathematics end up having surprising consequences]

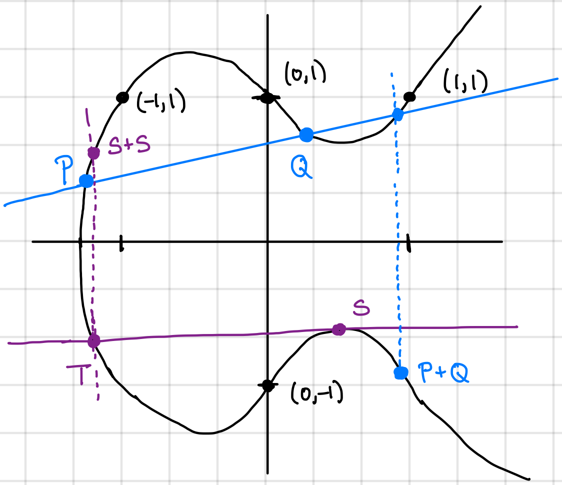
An elliptic curve is defined by an equation of the following form:

$$E: y^2 = x^3 + Ax + B \qquad \text{[we will assume that } \underbrace{4A^3 + 27B^2}_{\text{"discriminant" of the curve}} \neq 0]$$

non-zero to ensure there are no repeated roots (and the group law is well-defined)

where A, B are constants (over $\mathbb{R}$ or $\mathbb{C}$ or $\mathbb{Q}$ or $\mathbb{F}_p$)

Example of an elliptic curve: $y^2 = x^3 - x + 1$ (over the reals)



Consider the set of rational points on this curve

points where x- and y-coordinates are rational values

e.g., $(0, \pm 1)$, $(1, \pm 1)$, $(-1, \pm 1)$ [are there other points?]

Surprising facts:

1. Take any two rational points on the curve and consider the line that passes through them. The line will intersect the curve at a new point, which will also have rational coefficients.
2. Take any rational point on the curve and consider the tangent line through that point. The line will intersect the curve at a new point, which will also have rational coefficients.

Thus, given two rational points, there is a way to generate a third rational point.
↳ In fact, this operation essentially defines a group law (but with following modifications):
1. We introduce a "point at infinity" (e.g., a horizontal line at $y = \infty$), denote $\mathcal{O}$ (this is the identity element)
2. The group operation (called the "chord and tangent" method) maps two curve points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ to a point R by first computing the third point that along the line connecting P, Q and reflecting the point about the x-axis. [Observe that the reflection ensures that $\mathcal{O}$ is the identity)

↳ Remarkably, this defines a group law on the rational points on the elliptic curve, and we can write down algebraic relations for computing the group law (somewhat messy but there is a closed form expression)

In cryptography, we work over finite domains, so we instead consider elliptic curves over finite fields (rather than $\mathbb{R}$ or $\mathbb{C}$). Specifically, we write

$$E(\mathbb{F}_p) = \{x, y \in \mathbb{F}_p : y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}$$

No geometric interpretation of the group law over $\mathbb{F}_p$ (instead, define it using the algebraic definitions derived above)
↳ $E(\mathbb{F}_p)$ still forms a group under this group law

How big is the group $E(\mathbb{F}_p)$?

Theorem (Hasse). Let E be an elliptic curve with coefficients in $\mathbb{F}_p$. Then

$$\left| |E(\mathbb{F}_p)| - (p+1) \right| \leq 2\sqrt{p}$$

Thus, number of points on $E(\mathbb{F}_p)$ is roughly $p \pm \sqrt{p}$

Thus, if we want a curve with roughly 2^{256} points (i.e., a group with $\approx 2^{256}$ elements), it suffices to take $p \sim 2^{256}$ (256-bit prime).

naively, can compute in $O(p)$ time but this is not efficient!

But for cryptographic operations, we also need to know the order of the group (and ideally, the order should be prime). Schoof shows how to efficiently compute the number of points on $E(\mathbb{F}_p)$ [in time $O(\log^6 p)$]

↳ In practice, we have a set of standard elliptic curves that almost everyone uses (e.g., P-256, P-384, Curve25519)

In an elliptic-curve group, best algorithm for discrete log are the generic ones (e.g., running time $O(\sqrt{p})$), so we can use 256-bit curves to achieve 128-bits of security — significantly better than working in $\mathbb{Z}_p^*$ (or over RSA groups)!

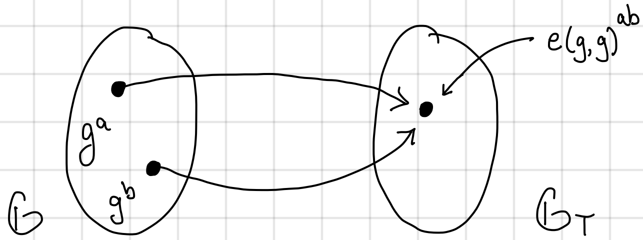
Another advantage of elliptic curves: they often support additional structure that can be leveraged in many cryptographic applications

↳ today, we will look at one specific example: pairing-based cryptography

prime order p

Definition. A (symmetric) pairing $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a mapping with the following properties: [also referred to as a bilinear map]

- Bilinearity: $\forall a, b \in \mathbb{Z}_p$, $g \in \mathbb{G}$: $e(g^a, g^b) = e(g,g)^{ab}$
- Non-degenerate: if g generates $\mathbb{G}$, then $e(g,g)$ generates $\mathbb{G}_T$
- Efficiency: there exists an efficient algorithm that evaluates the mapping e



Initial application of pairings was for attacking discrete log over elliptic curve groups: can map computing discrete log in $E(\mathbb{F}_p)$ to computing discrete logs in $\mathbb{F}_{p^\alpha}$ (for (hopefully) small α)

[algorithm due to Menezes, Okamoto, Vanstone, 1993] ↰ called the "embedding degree" of the elliptic curve

"Bug $\Longrightarrow$ Feature": $\underbrace{\text{[Joux, 2000] [Boneh, Franklin, 2001]}}_{\text{21st century cryptography!}}$

Application 1: 3-party non-interactive key exchange [Joux, 2000]

Recall classic Diffie-Hellman key exchange:

Alice $[a \xleftarrow{R} \mathbb{Z}_p]$ Bob $[b \xleftarrow{R} \mathbb{Z}_p]$

$$\xrightarrow{\quad g^a \quad}$$

$$\xleftarrow{\quad g^b \quad}$$

$\downarrow$ g^{ab} $\qquad$ $\downarrow$ g^{ab}

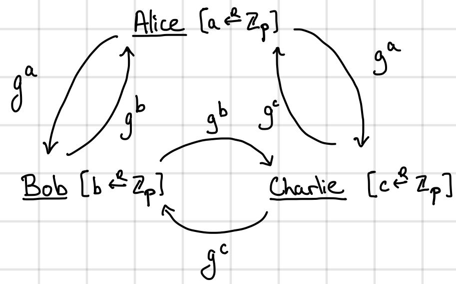
where r is random

Security: $(g^a, g^b, g^{ab}) \overset{c}{\approx} (g^a, g^b, g^r)$ by DDH

Essentially relies on discrete log being a "1-linear" map

[easy to compute linear relations in the exponent, but difficult to compute quadratic relations]

What if we had 3 parties? [Big open problem since Diffie-Hellman key exchange]



Everyone sees g^a, g^b, g^c $\Longrightarrow$ Alice can compute $e(g^b, g^c)^a$

Bob can compute $e(g^a, g^c)^b$

Charlie can compute $e(g^a, g^b)^c$

Shared key: $e(g,g)^{abc}$

Security: given g, g^a, g^b, g^c, require that $e(g,g)^{abc}$ looks indistinguable from random:

$$(g, g^a, g^b, g^c, e(g,g)^{abc}) \overset{c}{\approx} (g, g^a, g^b, g^c, g^r)$$ [Bilinear DDH (BDDH) assumption]

With a pairing, easy to compute quadratic relations in the exponent, but difficult to compute cubic relations in the exponent

Beyond 3 parties? Seems very difficult!

- Possible using multilinear maps ... several existing candidates, but all seem to be broken [cannot instantiate key-exchange directly]
- Also follows from indistinguishability obfuscation [security questionable and extremely far from being practical]

Even 3-party key exchange is not known from other assumptions [major open problem in lattice-based cryptography!]