Elliptic curves in cryptography with applications to digital asset ownership

Balázs Kőműves

Falkstenen AB

Budapest, 17 February 2014

Outline

This is an expository talk about three loosely connected subject:

- elliptic curves
- public-key cryptography
- digital money (in particular, bitcoin)

Some history

substitution ciphers	$\sim$500 BC	
"ROT13"	$\sim$70 BC	Julius Caesar
elliptic integrals	$\sim$ 1700–1750	Fagnano; Euler
elliptic functions	1829	Abel; Jacobi
Weierstrass $\wp$ function	1862	Weierstrass
Enigma machines	$\sim$ 1920–1950	
(more-or-less RSA)	(1973)	(Clifford Cocks et al)
public key cryptography	1976	Diffie & Hellmann
RSA	1977	Rivest, Shamir & Adleman
ElGamal cryptosystem	1984	ElGamal
Elliptic Curve Crypto	1985	Miller; Kolbitz
cryptographic hash functions	$\sim$ 1990	
bitcoin	2009	Satoshi Nakamoto

Elliptic curves

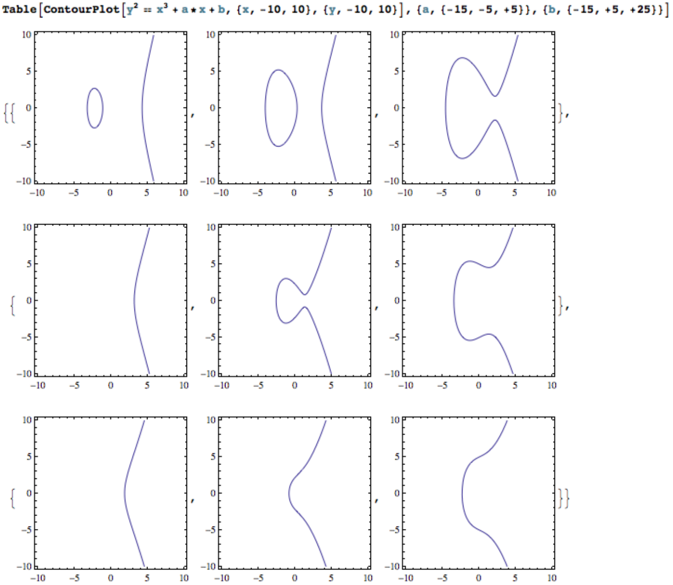
What is an *elliptic curve*?

- smooth cubic plane curve (with a base point)
- smooth genus 1 curve (with a base point)
- $y^2 = x^3 + ax + b$
  (“short Weierstrass form”; $\mathrm{char}(\mathbb{F}) \neq 2, 3$ and $4a^3 + 27b^2 \neq 0$)
- 1 dimensional abelian variety

Etimology / history:

- arc length of an ellipse
- elliptic integrals
- the inverse problem: elliptic functions
- elliptic curves

Pictures of elliptic curves over $\mathbb{R}$



```
Table[ContourPlot[y^2 == x^3 + a*x + b, {x, -10, 10}, {y, -10, 10}], {a, {-15, -5, +5}}, {b, {-15, +5, +25}}]
```

Arc length of an ellipse

An ellipse: $y = q\sqrt{1-x^2}$ with $q > 0$. The slope at (x, y):

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{-qx}{\sqrt{1-x^2}}$$

Let's write down the arc length:

$$\begin{aligned} S(t) &= \int_0^t \sqrt{\mathrm{d}x^2 + \mathrm{d}y^2} = \int_0^t \sqrt{1 + (\mathrm{d}y/\mathrm{d}x)^2}\,\mathrm{d}x = \\ &= \int_0^t \sqrt{\frac{1 - x^2 + q^2x^2}{1-x^2}}\,\mathrm{d}x \\ &=: E\big(t\,;\sqrt{1-q^2}\big) \end{aligned}$$

For $q = 1$ (circle), this simplifies; otherwise it is not an elementary function (not even for $t = 1$). $E(x; k)$ is called "incomplete elliptic integral of the second kind, Jacobi's form".

Generic elliptic integral: $\int R(x, \sqrt{P(x)})\mathrm{d}x$, where $P(x)$ is a cubic or quartic polynomial without double roots, and $R(x, y)$ is a rational function.

Weierstrass $\wp$ function

Let's fix two complex numbers $\omega_1, \omega_2 \in \mathbb{C}$ which generate the lattice

$$\Lambda = \{\, n\omega_1 + m\omega_2 \ : \ n, m \in \mathbb{Z} \,\} \subset \mathbb{C}.$$

An *elliptic function* is a meromorphic function which is periodic wrt. the lattice Λ.

Let us use $\Lambda_o = \Lambda \backslash \{0\}$ for brevity. The Weierstrass $\wp$ function is defined by

$$\wp(z) = \frac{1}{z^2} + \sum_{\omega \in \Lambda_o} \left[\frac{1}{(z-\omega)^2} - \frac{1}{\omega^2} \right]$$

It is meromorphic function, clearly periodic wrt. the lattice Λ (thus an elliptic function), an *even* function: $\wp(z) = \wp(-z)$, and has second order poles exactly at the points of Λ.

Fact: $\wp(z)$ is the *universal* elliptic function: Any elliptic function is a rational function of $\wp(z)$ and $\wp'(z)$.

The differential equation

Introduce the quantities:

$$g_2 = 60 \sum_{\omega \in \Lambda_o} \omega^{-4}$$
$$g_3 = 140 \sum_{\omega \in \Lambda_o} \omega^{-6}$$

The Laurent series expansion of $\wp(z)$:

$$\wp(z) = z^{-2} + \frac{g_2}{20} z^2 + \frac{g_3}{28} z^4 + O(z^6)$$

Theorem: The Weierstrass $\wp$ function satisfies the following differential equation:

$$[\wp'(z)]^2 = 4\,\wp(z)^3 - g_2\,\wp(z) - g_3$$

Proof: Comparing the poles of the two sides, we can conclude that their difference is a periodic entire function, and thus a constant (which can be readily computed as 0).

Conclusion I. (inverse problem)

Integrating the differential equations, we can see that for the elliptic integral

$$u(y) = -\int_y^\infty \frac{\mathrm{d}s}{\sqrt{4s^3 - g_2 s - g_3}}$$

we have $y = \wp(u(y))$, thus

$$\wp = u^{-1}.$$

That is, the Weierstrass $\wp$ function is the inverse of this elliptic integral. "Proof":

$$(\wp^{-1})'(y) = \frac{1}{\wp'(\wp^{-1}(y))} = \frac{1}{\sqrt{4y^3 - g_2 y - g_3}}$$

Similarly, other elliptic functions solve the inverse problems for other types of elliptic integrals (hence the name).

Conclusion II. (elliptic curve)

From the differential equations, we can see directly that the mapping

$$\begin{array}{ccccc} \mathbb{C} & \to & \mathbb{C}/\Lambda & \to & \mathbb{P}^2 \\ z & \mapsto & z \bmod \Lambda & \mapsto & \left[4\wp(z) : 4\wp'(z) : 1\right] \end{array}$$

is well-defined (actually, an isomorphism) between the complex torus $\mathbb{C}/\Lambda$ and the elliptic curve

$$y^2 = x^3 + ax + b$$

with $a = -4g_2$ and $b = -16g_3$.

Moreover, $\Lambda \mapsto (g_2, g_3)$ is an isomorphism between the moduli space of lattices and the moduli space of elliptic curves (whatever that means...).

The group law on elliptic curves

The torus $\mathbb{C}/\Lambda$ is naturally a group (it inherits the complex addition), so it shouldn't be too surprising that elliptic curves also have a group structure. Actually it *is* rather surprising :)

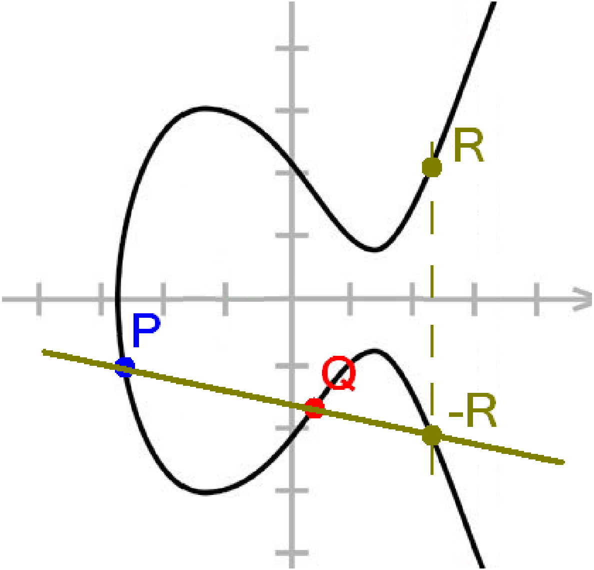
Definitions (for the Weierstrass form):

- identity element: The point at infinity (denoted by O)
- inverse: mirroring wrt. the X axis
- addition: if P, Q and R are on a straight line, we declare $P + Q + R = O$

Group laws:

- identity satisfies what it should (trivial)
- addition is commutative (trivial)
- addition is associative (nontrival!)

Addition on elliptic curves



Symmetric key cryptography

The two parties have a shared secret key; they can then encrypt and decrypt messages using this key.

With modern symmetric key encryption standards, it is infeasible for an attacker to guess the message without knowing the secret key.

The big issue: *How to agree on a secret key?* In practice, this needs meeting in person (often inpractical) and at a secure location (can be inpractical, or even impossible). Also, we want machines to communicate safely, too.

Nevertheless, symmetric key encryption algorithms are useful components of larger crypto systems.

Asymmetric (or public) key cryptography

Each party has a *pair* of corresponding keys: one which is public (say, on published on their homepage) and one which is private (only a single person knows it).

Main applications:

- Estabilishing a shared secret without meeting (key exchange)
- Send messages which only the intended recepient can decrypt (encryption)
- Prove that a message was really written by the person who claims it (signature)

From these basic building blocks, a huge set of really interesting applications can be built.

Public-key crypto is widely used on the internet today: HTTPS, SSL, PGP, Bitcoin, etc...

Public key cryptography, II.

Public-key cryptosystems are based on problems which are easy to compute in one direction, but hard to compute in the other direction:

- factorization of the product of two large prime numbers (RSA)
- discrete logarithm (ElGamal)
- elliptic curve discrete logarithm (ECC)

Discrete logarithm: Fix a finite cyclic group $\mathcal{G}$ of order n and a generator $G \in \mathcal{G}$.

- private key: a random number $d \in [1, n-1] \subset \mathbb{N}$
- public key: the group element $Q = dG \in \mathcal{G}$

The idea is that it is very hard to determine d from Q (for appropriate choices of $\mathcal{G}$). Elliptic curve crypto: Let $\mathcal{G}$ be an appropriately chosen elliptic curve over a finite field $\mathbb{F}_q$.

Elliptic curves over finite fields

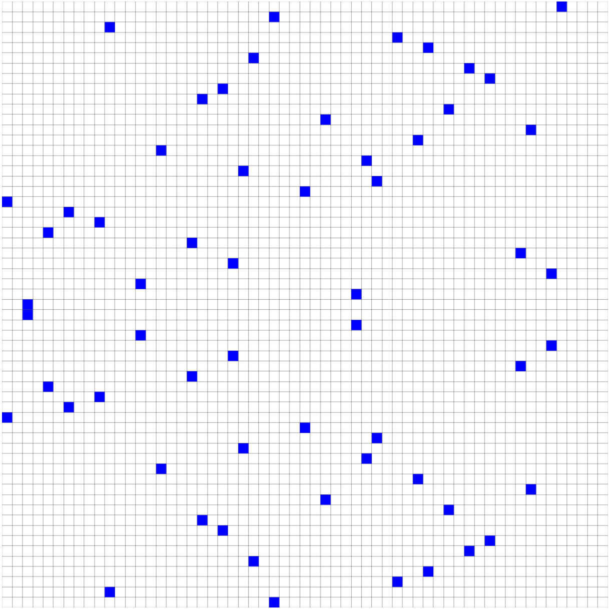
Elliptic curves make sense over different fields: $\mathbb{C}$, $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{F}_q$, etc. To be able to do cryptography, we need computable objects; thus the choice of finite fields $\mathbb{F}_q$. In practice either $q = 2^m$ or a prime $q = p$.

The safety of ECC depends on the hardness of the elliptic curve discrete logarithm problem. Not all curves are created equal!

An example, here is the standardized curve called `secp256k1` (this is the curve used by Bitcoin):

- the field is $\mathbb{F}_p$ with $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
- the curve equation is $y^2 = x^3 + 7$ (that is, $a = 0$ and $b = 7$)
- the number of points on the curve n is also a prime, and "close" to p (to be more precise, $p - 2^{129} < n < p - 2^{128}$)
- since n is prime, the group is cyclic; thus any element (except the infinity) will do as the generator G
- but there is a concrete, randomly-looking G in the standard

A picture of $y^2 = x^3 + 7$ over $\mathbb{F}_{59}$



Remark: $p \approx 2^{256} \approx 10^{77} \approx$ no. of elementary particles in the universe

Key length comparison

The primary advantage of ECC is that we can have shorter keys for equivalent security:

Security level	RSA key length	ECC key length	ratio
80	1024	160-223	5–6
112	2048	224-255	8–9
128	3072	256-283	11–12
192	7680	384-511	15–20
256	15360	512-571	27–30

The shorter key length can be actually important in practice:

- embedded devices with limited resources (smartcards, etc)
- large number of signatures: both bandwidth and storage can be important (bitcoin)

Diffie-Hellmann key exchange

Recall that a *key pair* (d, Q) consists of

- a private key, which is a random number $d \in [1, n-1] \subset \mathbb{N}$
- a public key, which is the group element $Q = dG \in \mathcal{G}$

Let there be two parties: Alice and Bob, with key pairs (d_A, Q_A) and (d_B, Q_B). They can compute a shared secret $S \in \mathcal{G}$ as follows:

$$d_A Q_B = d_A(d_B G) = \underbrace{(d_A d_B)G}_{S} = d_B(d_A G) = d_B Q_A$$

Alice can knows d_A, so she can compute the leftmost version; Bob knows d_B, so he can compute the rightmost version. But nobody else knows neither d_A or d_B, thus S is their secret.

They can then proceed and use S for any purpose, for example as the key of a symmetric encryption scheme.

Public-key encryption

Alice wants to send a message to Bob, but wants to make sure that nobody else can read it.

This can be implemented as an application of the Diffie-Hellmann key exchange:

1. Alice generates an ephemeral key pair (d_E, Q_E)
2. she then computes a shared secret $S = d_E Q_B$
3. computes a symmetric key $k = k(S)$ from S
4. encrypts the message m with the key k
5. sends Q_E and the ciphertext $c_k(m)$ to Bob

On the other side: Bob computes $S = d_B Q_E$, then $k = k(S)$, and decrypts the message. Nobody else knows neither d_E or d_B.

In practice it is a bit more complicated, but that's the idea.

Intermezzo - Hash functions

A *hash function* H is a function

$$H : \bigcup_{k=0}^{\infty} \{0,1\}^k \to \{0,1\}^m$$

with a fixed m (usually $m \in \{32, 64, 128, 160, 256, 512\}$), such that the output looks more-or-less random, and changing any single bit of the input completely changes the output.

A hash function is called *cryptographic*, if it is practically impossible for any given hash h to figure out an input x such that $H(x) = h$, unless we already know x ("preimage resistance"); it is also desirable to be impossible to find two inputs $x_1 \neq x_2$ with $H(x_1) = H(x_2)$ ("collision resistance").

Hash functions (both cryptographic and non-cryptographic) are very widely used in computer science. For cryptographic hash functions there are international standards, for example the SHA2 family.

Elliptic Curve Digital Signature Algorithm

Alice writes a message m, and wants to prove that she wrote it. She already has key pair (d_A, Q_A), and people accept that the public key Q_A in fact belongs to her.

Construction of the signature:

1. compute a hash $z = \mathrm{HASH}(m) \in [1, n-1]$ of the message m
2. generate an ephemeral key pair: k and $Q_k = kG = (x, y)$
3. let $r = (x \bmod n) \in \mathbb{Z}_n$
4. let $s = k^{-1}(z + rd_A) \in \mathbb{Z}_n$
5. the signatures is $(r, s) \in \mathbb{Z}_n \times \mathbb{Z}_n$

Verification of the signature:

1. compute $z = \mathrm{HASH}(m) \in [1, n-1]$ as before
2. compute $u = s^{-1}z \in \mathbb{Z}_n$ and $v = s^{-1}r \in \mathbb{Z}_n$
3. compute the curve point $(x, y) = Q = uG + vQ_A \in \mathcal{G}$
4. the signature is valid iff $x = r$.

Representations of elliptic curve points

There are many different representations of elliptic curves:

- Weierstrass affine coordinates $\mathbb{A}^2$
- Weierstrass projective coordinates $\mathbb{P}^2$
- Weierstrass weighted projective coordinates $\mathbb{P}(2, 3, 1)$
- Montgomery form
- Hessian form
- Jacobian form
- Edwards form
- etc...

Representation matters because of efficiency! And possibly also different security properties.

Computations with elliptic curves, I.

How to compute $dQ \in \mathcal{G}$ efficiently, with $d \in \mathbb{Z}_n$ and $Q \in \mathcal{G}$?

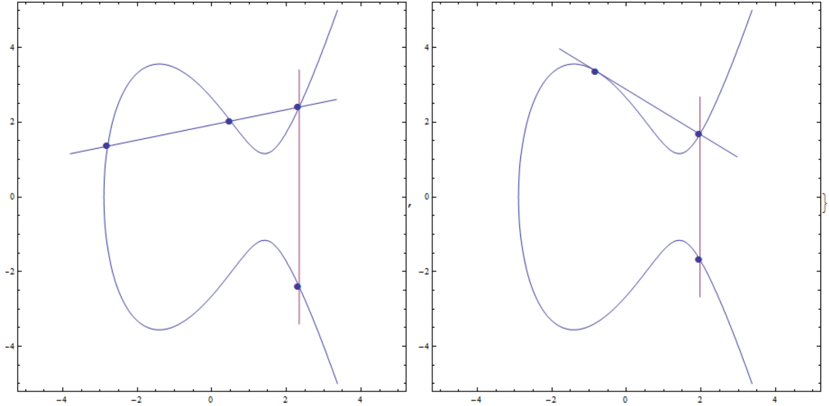
Answer: "fast exponentation"! Write d in binary form:
$d = \sum_{i=0}^{m-1} d_i 2^i$, where $d_i \in \{0, 1\}$.

$$dQ = \left(\sum_{i=0}^{m-1} d_i 2^i\right) \cdot Q = \sum_{i=0}^{m-1} d_i (2^i Q) = \sum_{i=0}^{m-1} d_i Q_i$$

where $Q_i = 2^i Q$ can be computated by *repeated doubling*:
$Q_0 = Q,\ \ Q_1 = 2Q_0,\ \ Q_2 = 2Q_1,\ \ Q_3 = 2Q_2$, etc...

Thus we need addition and doubling (which is a special case of addition, but needs to be handled separately anyway).

Elliptic curve addition and doubling in pictures



Elliptic curve addition and doubling in Weierstrass form

In any field $\mathbb{F}$ $(\mathrm{char}(\mathbb{F}) \neq 2, 3)$, for two points $P \neq Q \neq O$ on the elliptic curve $y^2 = x^3 + ax + b$, with coordinates $P = (x_p, y_p)$ and $Q = (x_q, y_q)$, it is straightforward (?) to calculate the coordinates of $P + Q = R = (x_r, y_r)$ and $2P = U = (x_u, y_u)$:

$$\begin{aligned} s &= \tfrac{y_q - y_p}{x_q - x_p} & \qquad t &= \tfrac{3x_p^2 + a}{2y} \\ x_r &= s^2 - (x_p + x_q) & x_u &= t^2 - 2x_u \\ y_r &= -y_p - s(x_r - x_p) & y_u &= -y_p - s(x_u - x_p) \end{aligned}$$

Here s resp. t are the slopes of the secant resp. tangent lines.

Projective coordinates

The problem: Each addition or doubling needs a division in $\mathbb{F}_q$, which is *slow*; each exponentiation needs a *lot* of additions and doublings; and we need several exponentiation to do cryptography.

The divisions are the bottleneck. How to make divisions faster? Answer: Don't do divisions!

Using projective coordinates, we only need one division at the end, when we convert back to affine coordinates. Using weighted projective coordinates $\mathbb{P}(2, 3, 1)$ is even better (somewhat less multiplications/additions):

$$[x : y : z] = [\lambda^2 x : \lambda^3 y : \lambda z] \in \mathbb{P}(2, 3, 1).$$

Other representations can be even more efficient.

Digital assets

Physical objects exist in single copies, thus ownership is more-or-less clear. In contrast, digital objects (information) can be readily copied in any number of copies. So how can we ensure that a digital asset is owned by a single entity at a given point of time?

- centralized database of ownership (eg. banks, clearing houses)
  - requires trust in the central authority
  - single point of failure
  - currently rather expensive (high fees)
- decentralized ledger
  - how do people agree on what exactly is in the ledger?
  - how to prevent double-spending?

Bitcoin's solution:

- digitally signed transactions
- consensus on the ledger via vote-by-CPU-power

Bitcoin

Bitcoin is a decentralized digital currency protocol.

- bitcoin *addresses* correspond to private/public key pairs (more precisely, addresses are hashes of public keys)
- addresses can hold amounts of “coins”
- transactions “spend” the output(s) of previous transaction(s)
- a transaction specifies how its output(s) can be spend in the future
  - typical transcation: send the coins to one or more address(es)
  - the spending condition: the next spender(s) must prove that they own the private key(s) corresponding to these addreses
- transactions are stored in the public ledger (which is a decentralized database)
- it’s the the set of transactions which is fundamental - the “coins” are just invariants.

Bitcoin transaction example, part I.

I expect income from two sources:

- I sold a book to a Alice for 0.3 BTC
- I made a bet with Bob, and won 0.2 BTC

I generate two key-pairs $(d_{1,2}, Q_{1,2})$ and derive the corresponding addresses $A_{1,2}$. Give the address A_1 to Alice and the address A_2 to Bob. They send me the agreed amounts. Two transactions $T_{1,2}$ will appear in the ledger:

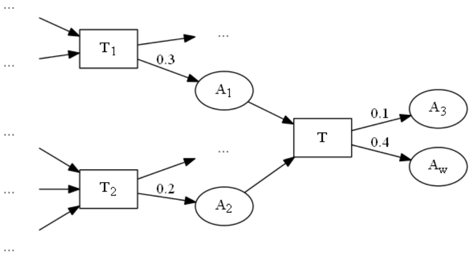
- T_1 says that 0.3 BTC can be spent by anybody who can prove that they know d_1
- T_2 says that 0.2 BTC can be spent by anybody who can prove that they know d_2

This means that I “own” 0.5 BTC alltogether (since nobody else knows $d_{1,2}$).

Next, let’s say I want to buy a bottle of whisky for 0.4 BTC!

Bitcoin transaction example, part II.

To send 0.4 BTC for the address of the whisky shop A_w, I need to craft a transaction which "spends" the outputs of the previous transactions T_1 and T_2, sends 0.4 BTC of it to the whisky shop, and sends the remaining 0.1 BTC back to me (usually to a freshly generated address A_3)



The transaction T is signed with the private keys corresponding to the addresses A_1 and A_2, to prove that they belong to me.