

Fast, uniform scalar multiplication for  
genus 2 Jacobians with fast Kummers  
*and*  
 $\mu$ Kummer: efficient hyperelliptic signatures  
and key exchange on microcontrollers

Ping Ngai (Brian) Chung   Craig Costello   **Benjamin Smith**  
Joost Renes   Peter Schwabe   Lejla Batina

University of Chicago   Microsoft Research

**INRIA + Laboratoire d'Informatique de l'École polytechnique (LIX)**

Radboud Universiteit Nijmegen

ECC 20 :: Izmir, Turkey :: 06/09/2016

## Genus 2 cryptography

Genus 2 is an alternative to elliptic curves (genus 1) as a source of groups for DLP-based cryptosystems.

So far: best DLP algorithm is Pollard  $\rho$

$\implies$  same DLP hardness per bit as elliptic curves

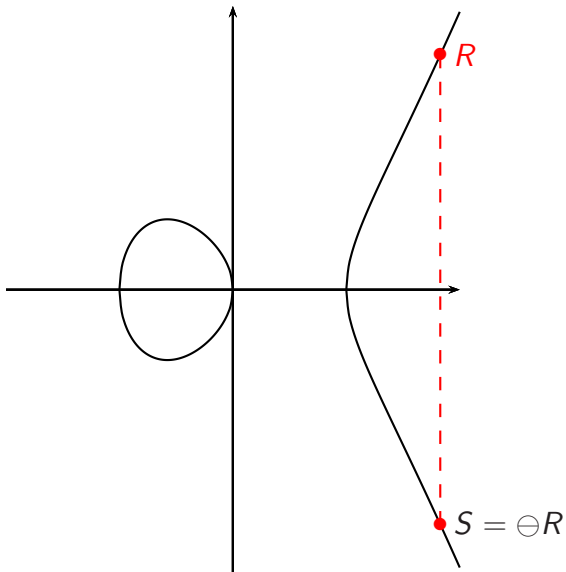
$\implies$  same size keys (optimally small).

*Why bother?*

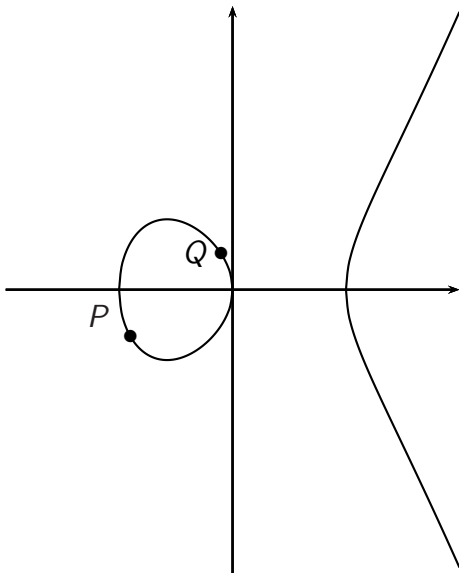
Sometimes genus 2 can offer better performance.

Plus, it's much more fun.

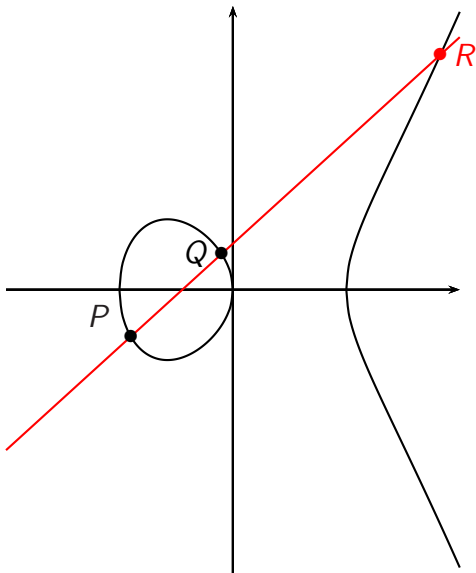
Elliptic curve negation:  $\ominus R = S$



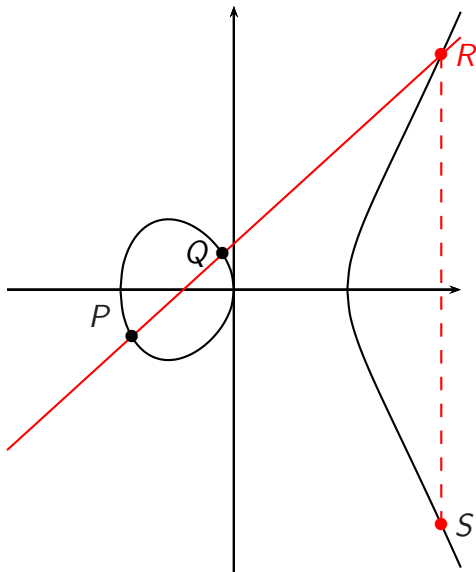
Elliptic curve addition:  $P \oplus Q = ?$



Elliptic curve addition:  $P \oplus Q \oplus R = 0$

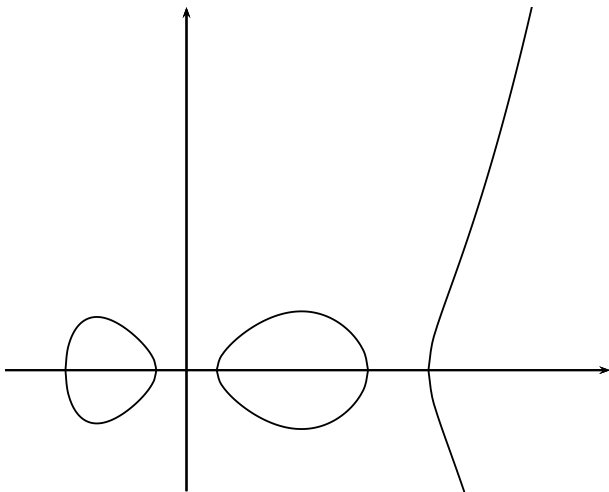


Elliptic curve addition:  $P \oplus Q = \ominus R = S$



## Genus 2 curves

$C : y^2 = f(x)$  with  $f \in \mathbb{F}_p[x]$  degree 5 or 6 and squarefree



## Making groups from genus 2 curves

---

*Jacobian*: algebraic group  $\mathcal{J}_C \sim \mathcal{C}^{(2)}$ :

Elements are **pairs of points** on  $\mathcal{C} : y^2 = f(x)$ ,  
with all pairs  $\{(x, y), (x, -y)\}$  “blown down” to 0.

Negation  $\ominus : \{(x_1, y_1), (x_2, y_2)\} \mapsto \{(x_1, -y_1), (x_2, -y_2)\}$

Group law on  $\mathcal{J}_C$  induced by

$$\{P_1, P_2\} \oplus \{Q_1, Q_2\} \oplus \{R_1, R_2\} = 0$$

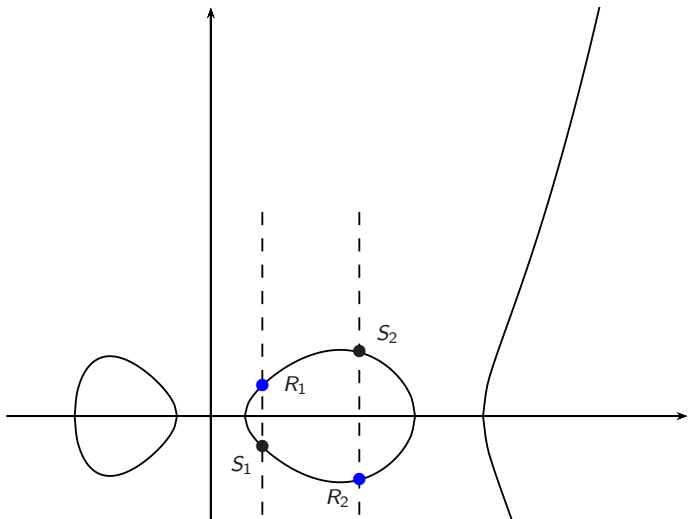
whenever  $P_1, P_2, Q_1, Q_2, R_1, R_2$  are  
the intersection of  $\mathcal{C}$  with some cubic  $y = g(x)$ .

*Why? Any 4 points in the plane determine a cubic  $y = g(x)$ , which must intersect  $\mathcal{C} : y^2 = f(x)$  in 6 points because  $g(x)^2 = f(x)$  has 6 solutions.*

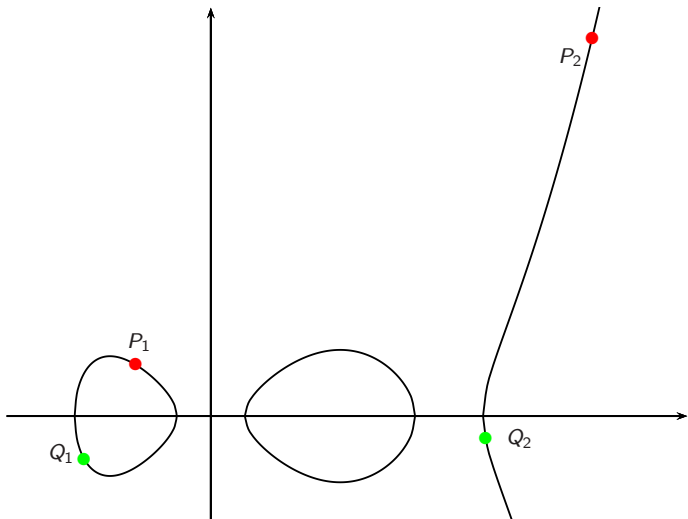


Genus 2 group law: negation  $\ominus\{R_1, R_2\} = \{S_1, S_2\}$

---

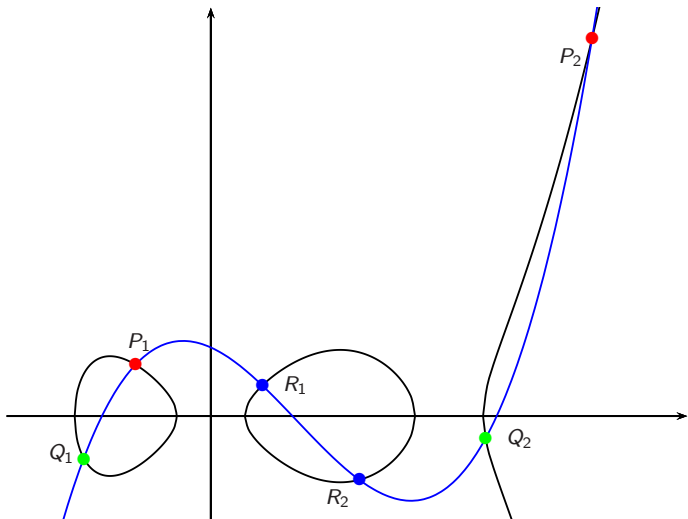


*Genus 2 group law:  $\{P_1, P_2\} \oplus \{Q_1, Q_2\} = ?$*

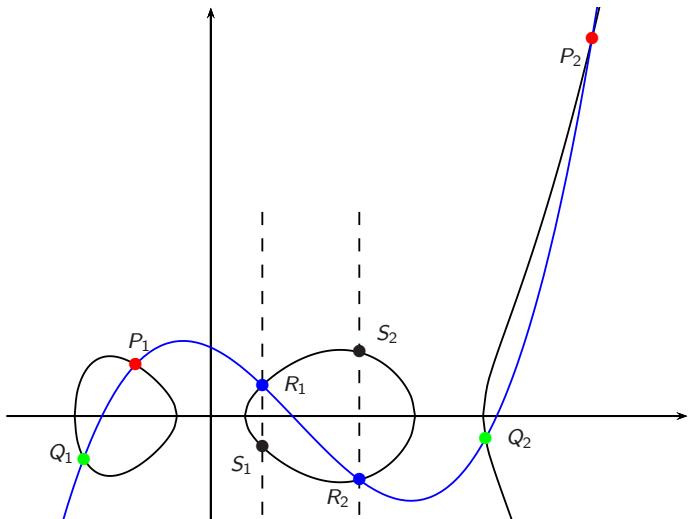


Genus 2 group law:  $\{P_1, P_2\} \oplus \{Q_1, Q_2\} \oplus \{R_1, R_2\} = 0$

---



Genus 2 group law:  $\{P_1, P_2\} \oplus \{Q_1, Q_2\} = \ominus\{R_1, R_2\} = \{S_1, S_2\}$



We (still) want to implement basic cryptosystems based on Discrete Log and Diffie–Hellman problems in  $\mathcal{J}_C(\mathbb{F}_p)$

Protocols like Diffie–Hellman key exchange,  
Schnorr signatures, (EC)DSA signatures:

We *occasionally* need to compute  $P \oplus Q$ .

We *mostly* need to compute  $[m]P$ .

Side channel safety  $\implies$  scalar multiplication must be *uniform* and *constant-time* when the scalar  $m$  is secret.

# Schnorr signatures: signing

Fixed public base point  $P$  in a  $2t$ -bit group;  $N = \#\langle P \rangle$ .

Keys: private  $x$ , public  $Q = [x]P$ .

To sign a message  $m$  with the key pair  $(Q, x)$ :

---

## Algorithm 1 Schnorr signature signing operation

---

- 1: **function** SIGN( $m \in \{0, 1\}^*$ ,  $x \in \mathbb{Z}/N\mathbb{Z}$ )
  - 2:      $r \leftarrow \text{random}(\mathbb{Z}/N\mathbb{Z})$
  - 3:      $R \leftarrow [r]P$  ▷ mult. public point by secret scalar
  - 4:      $e \leftarrow H(m||R)$  ▷  $H$ : hash function,  $t$  bit output
  - 5:      $s \leftarrow r - ex \pmod{N}$  ▷ (so  $[s]P \oplus [e]Q = R$ )
  - 6:     **return**  $(s, e) \in (\mathbb{Z}/N\mathbb{Z}) \times [0..2^t]$  ▷  $3t$  bits of storage
  - 7: **end function**
- 

$\implies$  signatures at the 128-bit security level:

Elliptic and genus 2 Schnorr require 384 bits, RSA requires 3072.

## Schnorr signatures: verification

To verify a claimed signature  $(m, e)$  on a message  $m$  against a public key  $Q$ ,

---

### Algorithm 2 Schnorr signature verification

---

- 1: **function** VERIFY( $(s, e), m, Q$ )
  - 2:      $R' \leftarrow [s]P \oplus [e]Q$                    ▷ mult. public points, scalars
  - 3:      $e' \leftarrow H(m||R')$
  - 4:     **return**  $e' = e$
  - 5: **end function**
- 

DLP hardness + hash preimage hardness  
gives  $t$  bits of authenticity, integrity, and non-repudiability.

## ...So you want to instantiate a DLP/DHP-based protocol

---

Smallest key size for a given security level:  
use an *elliptic curve* or a *genus 2 Jacobian*.

### *For signatures and encryption:*

Elliptic: Edwards curves (eg. Ed25519), NIST curves, etc.

Genus 2: Jacobian surfaces.

Scalar mult: *Uniform* genus 2 is **much slower** than elliptic curves.

### *For Diffie–Hellman:*

Elliptic:  $x$ -lines of Montgomery curves (eg. Curve25519)

Genus 2: Kummer surfaces (Jacobians modulo  $\pm 1$ ).

Scalar mult: *Uniform* genus 2 can be **faster** than elliptic curves.

*E.g.: Bos–Costello–Hisil–Lauter (2013+14),*

*Bernstein–Chuengsatiansup–Lange–Schwabe (Asiacrypt 2014)*



## Why is uniform genus 2 tricky?

Elements  $\{P_1, P_2\}$ : separate, *incompatible* representations for cases where one or both of the  $P_i$  are at infinity.

Group law  $\{P_1, P_2\} \oplus \{Q_1, Q_2\} = \{S_1, S_2\}$ :

branch-tacular, separate special cases for  $P_i, Q_i$  at infinity, for  $P_i = P_j$ , for  $P_i = Q_j$ , for  $\{P_1, P_2\} = \{Q_1, Q_2\}, \dots$

These special cases are never implemented in “record-breaking” genus 2 implementations, but they’re easy to attack in practice.

For elliptic curves, we can always sweep the special cases under a convenient line to get a uniform group law, but in genus 2 this is much harder; *protection kills performance*.

## Why is Diffie–Hellman different?

Now you know why genus 2 Jacobians are painful candidates for cryptographic groups in practice.

So why is genus 2 fast and safe for Diffie–Hellman?

Because DH *doesn't need a group law*,  
just scalar multiplication;

so we can “drop signs” and work modulo  $\ominus$ .

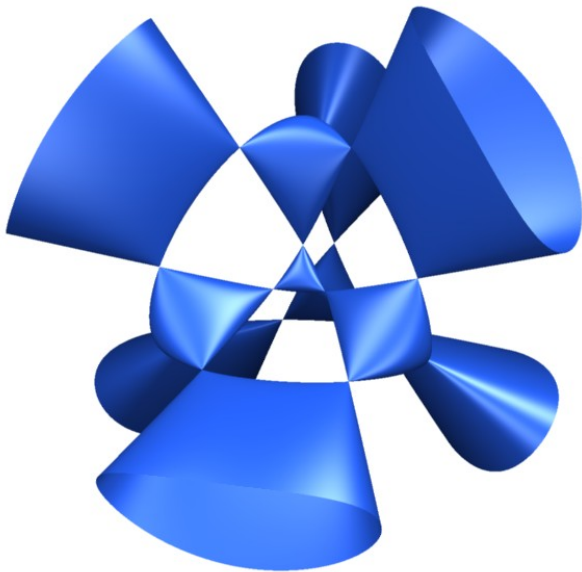
Alice computes  $(a, \pm P) \mapsto \pm[a]P$ ; Bob  $(b, \pm[a]P) \mapsto \pm[ab]P \dots$

Elliptic curves: work on  $x$ -line  $\mathbb{P}^1 = \mathcal{E}/\langle \pm 1 \rangle$

*Eg. Curve25519 (Bernstein 2006).*

Genus 2: work on the **Kummer surface**  $\mathcal{K}_c := \mathcal{J}_c/\langle \pm 1 \rangle$ .

## What a Kummer surface looks like



## Moving from $\mathcal{J}_C$ to the Kummer $\mathcal{K}_C$

Quotient map  $\mathbf{x} : \mathcal{J}_C \longrightarrow \mathcal{K}_C$  (ie  $\mathbf{x}(P) = \pm P$ )

No group law on  $\mathcal{K}_C$ :  $\mathbf{x}(P)$  and  $\mathbf{x}(Q)$  determines  $\mathbf{x}(P \oplus Q)$  and  $\mathbf{x}(P \ominus Q)$ , but we can't tell which is which.

Still,  $\ominus[m](P) = [m](\ominus P)$  for any  $m \in \mathbb{Z}$  and  $P \in \mathcal{J}_C$ ,  
so we do have a “scalar multiplication” on  $\mathcal{K}_C$ :

$$[m] : \mathbf{x}(P) \longmapsto \mathbf{x}([m]P) .$$

**Problem:** How do we compute  $[m]$  efficiently, *without*  $\oplus$ ?

Any 3 of  $\mathbf{x}(P)$ ,  $\mathbf{x}(Q)$ ,  $\mathbf{x}(P \ominus Q)$ , and  $\mathbf{x}(P \oplus Q)$  determines the 4th, so we can define

*pseudo-addition*

$$\mathbf{xADD} : (\mathbf{x}(P), \mathbf{x}(Q), \mathbf{x}(P \ominus Q)) \mapsto \mathbf{x}(P \oplus Q)$$

*pseudo-doubling*

$$\mathbf{xDBL} : \mathbf{x}(P) \mapsto \mathbf{x}([2]P)$$

Bonus: easier to hide/avoid special cases in  $\mathbf{xADD}$  than  $\oplus$ .

## Kummer surface arithmetic

Let  $0_K$  be the image of  $0_{\mathcal{J}_C}$  in  $\mathcal{K}_C$ , and define

$$\mathcal{M} : ((x_1 : y_1 : z_1 : t_1), (x_2 : y_2 : z_2 : t_2)) \mapsto (x_1 x_2 : y_1 y_2 : z_1 z_2 : t_1 t_2)$$

$$\mathcal{S} : (x : y : z : t) \mapsto (x^2 : y^2 : z^2 : t^2)$$

$$\mathcal{I} : (x : y : z : t) \mapsto (1/x : 1/y : 1/z : 1/t)$$

$$\mathcal{H} : (x : y : z : t) \mapsto (x' : y' : z' : t') \text{ where } \begin{cases} x' = x + y + z + t, \\ y' = x + y - z - t, \\ z' = x - y + z - t, \\ t' = x - y - z + t. \end{cases}$$

- $\mathbf{xDBL}(\mathbf{x}(P)) = \mathcal{M}(\mathcal{HM}(\mathcal{S}(\mathcal{HS}(\mathbf{x}(P)))), \mathcal{IH}(0_K)), \mathcal{I}(0_K))$
- $\mathbf{xADD}(\mathbf{x}(P), \mathbf{x}(Q), \mathbf{x}(P \ominus Q))$   
 $= \mathcal{M}(\mathcal{HM}(\mathcal{M}(\mathcal{HS}(\mathbf{x}(P))), \mathcal{HS}(\mathbf{x}(Q))), \mathcal{IH}(0_K)), \mathcal{I}(\mathbf{x}(P \ominus Q)))$

The *green things* here are constants, or constants in practice.

Evaluate  $[m]$  by combining **xADDs** and **xDBLs**  
 using **differential** addition chains  
*(ie. every  $\oplus$  has summands with known difference).*  
 Classic example: the Montgomery ladder.

---

**Algorithm 3** The Montgomery ladder in a group

---

```

1: function LADDER( $m = \sum_{i=0}^{\beta-1} m_i 2^i, P$ )
2:    $(R_0, R_1) \leftarrow (0, P)$ 
3:   for  $i := \beta - 1$  down to 0 do
4:      $(R_{m_i}, R_{-m_i}) \leftarrow ([2]R_{m_i}, R_{m_i} \oplus R_{-m_i})$ 
5:   end for           ▷ invariant:  $(R_0, R_1) = ([\lfloor m/2^i \rfloor]P, [\lfloor m/2^i \rfloor + 1]P)$ 
6:   return  $R_0$            ▷  $R_0 = [m]P, R_1 = [m]P \oplus P$ 
7: end function

```

---

For each group operation  $R_0 \oplus R_1$ , **the difference  $R_0 \ominus R_1$  is fixed**  
 $\implies$  trivial adaptation from  $\mathcal{J}_c$  to  $\mathcal{K}_c$

---

**Algorithm 4** The Montgomery ladder on the Kummer

---

```
1: function LADDER( $m = \sum_{i=0}^{\beta-1} m_i 2^i$ ,  $\mathbf{x}(P)$ )
2:    $(x_0, x_1) \leftarrow (\mathbf{x}(0), \mathbf{x}(P))$ 
3:   for  $i := \beta - 1$  down to 0 do
4:      $(x_{m_i}, x_{-m_i}) \leftarrow (\mathbf{xDBL}(x_{m_i}), \mathbf{xADD}(x_0, x_1, \mathbf{x}(P)))$ 
5:   end for    $\triangleright$  invariant:  $x_0 = \mathbf{x}(\lfloor m/2^i \rfloor P)$ ,  $x_1 = \mathbf{x}(\lfloor m/2^i \rfloor + 1)P$ 
6:   return  $x_0$  ( $= \mathbf{x}(\lfloor m \rfloor P)$ )
7: end function
```

---

High symmetry of  $\mathcal{K}_c$

$\implies$  fast, vectorizable  $\mathbf{xADD}$  and  $\mathbf{xDBL}$  (Gaudry)

$\implies$  very fast Kummer-based Diffie–Hellman implementations

Eg. Bos–Costello–Hisil–Lauter (2013/14),

Bernstein–Chuengsatiansup–Lange–Schwabe (Asiacrypt 2014).



## Pulling a $y$ -rabbit out of an $x$ -hat

Kummer multiplication computes  $\mathbf{x}([m]P)$  from  $\mathbf{x}(P)$   
—but we need  $[m]P$  for signatures...

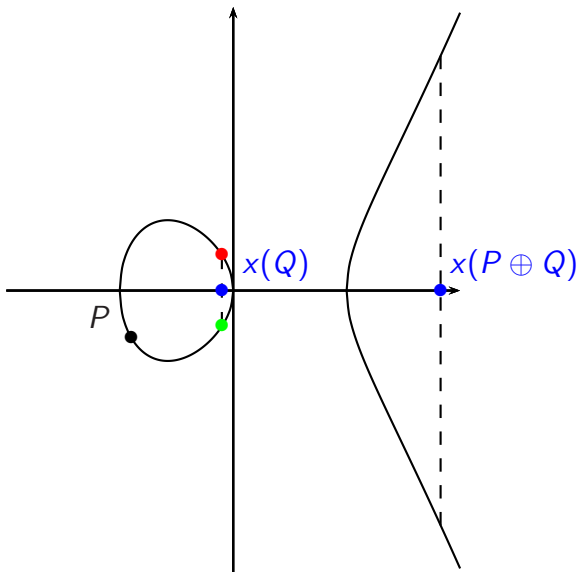
Mathematically, we threw away the sign:  
you can't deduce  $[m]P$  from  $P$  and  $\mathbf{x}([m]P)$ .

But there's a trick: if you computed  $\mathbf{x}([m]P)$   
using the Montgomery ladder, then you can!

At the end of the loop,  $x_0 = \mathbf{x}([m]P)$  and  $x_1 = \mathbf{x}([m]P \oplus P)$ ;  
and  $P$ ,  $\mathbf{x}(Q)$ , and  $\mathbf{x}(Q \oplus P)$  uniquely determines  $Q$  (for any  $Q$ ).

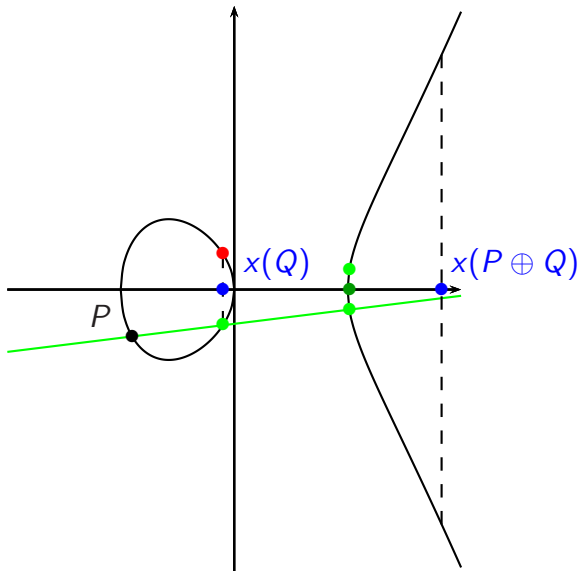
This is an old trick for elliptic curves: cf. López–Dahab (CHES 99),  
Okeya–Sakurai (CHES 01), Brier–Joye (PKC 02).

Given  $P$ ,  $x(Q)$ , and  $x(P \oplus Q)$ . Is  $Q = \bullet$  or  $\bullet$ ?

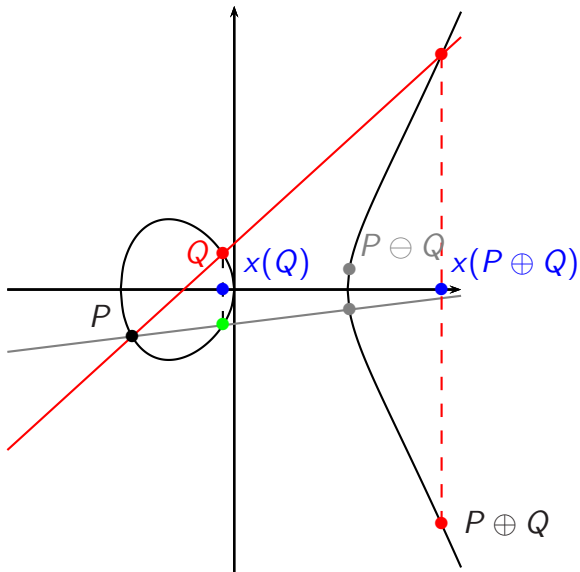


Line through  $P$  and  $\bullet$ : "sum" incompatible with  $\mathbf{x}(P \oplus Q)$ .

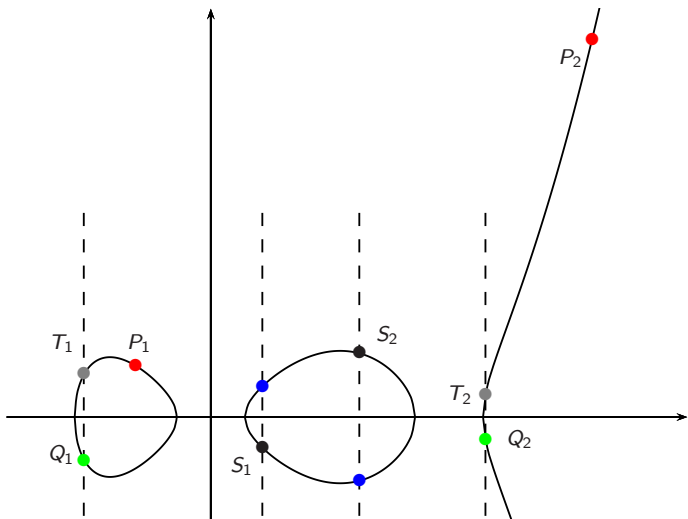
---



Line through  $P$  and  $\bullet$ : "sum" fits with  $x(P \oplus Q)$ , so  $Q = \bullet$

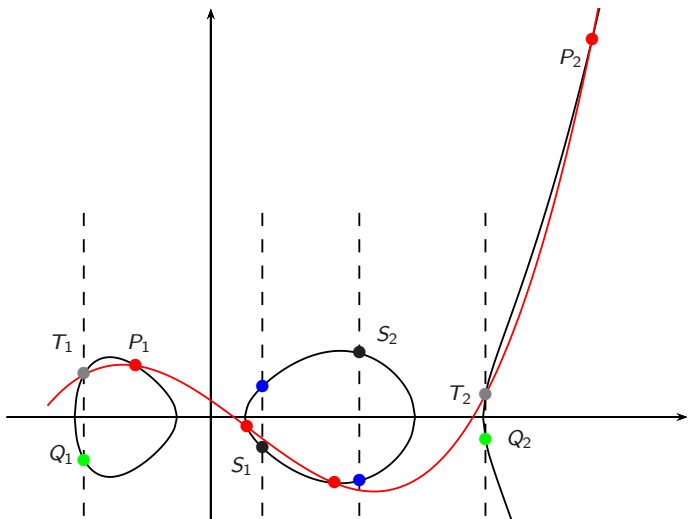


Genus 2 group law:  $\{P_1, P_2\} \oplus \{Q_1, Q_2\} = \{S_1, S_2\}$



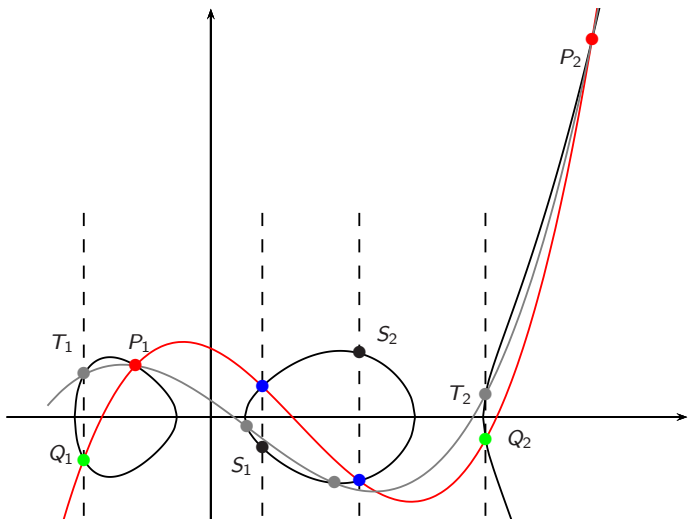
We know  $\{P_1, P_2\}, \mathbf{x}(\{Q_1, Q_2\}) = \mathbf{x}(\{T_1, T_2\})$ , and  $\mathbf{x}(\{S_1, S_2\}) = \mathbf{x}(\{R_1, R_2\})$ .  
 How can we find  $\{Q_1, Q_2\}$  (and not  $\{T_1, T_2\}$ )?

Genus 2 group law:  $\{P_1, P_2\} \oplus \{Q_1, Q_2\} = \{S_1, S_2\}$



Choosing  $\{T_1, T_2\}$  as (the wrong) preimage of  $\mathbf{x}(\{Q_1, Q_2\})$  yields a cubic **incompatible** with  $\mathbf{x}(\{S_1, S_2\})$ .

Genus 2 group law:  $\{P_1, P_2\} \oplus \{Q_1, Q_2\} = \{S_1, S_2\}$



There is only **one cubic** compatible with  $\{P_1, P_2\}$ ,  $x(\{Q_1, Q_2\})$ , and  $x(\{S_1, S_2\})$ .  
 It then intersects with  $\mathcal{C}$  in the (correct) preimage  $\{Q_1, Q_2\}$ .

Chung–Costello–S. (SAC 2016):

Recover algorithm implementing this trick for genus 2.

*Input:*  $P$  in  $\mathcal{J}_C(\mathbb{F}_q)$ ,  $\mathbf{x}(Q)$  and  $\mathbf{x}(P \oplus Q)$  in  $\mathcal{K}_C(\mathbb{F}_q)$   
(for some unknown  $Q$  in  $\mathcal{J}_C(\mathbb{F}_q)$ ).

*Output:*  $Q$  in  $\mathcal{J}_C(\mathbb{F}_q)$ .

Application: put

$(P, \mathbf{x}(Q), \mathbf{x}(P \oplus Q)) = (P, \mathbf{x}([m]P), \mathbf{x}([m+1]P))$   
(ie,  $P$  with output of  $\mathbf{x}$ -Montgomery ladder); then

$$\text{Recover}(P, \mathbf{x}([m]P), \mathbf{x}([m+1]P)) = [m]P .$$

$\implies$  1D (Montgomery) and 2D (Bernstein) SM.



## Chung–Costello–S. (SAC 2016):

---

### Algorithm 5 Montgomery/Kummer-based multiplication on the Jacobian

---

```
1: function SCALARMULTIPLY( $m = \sum_{i=0}^{\beta-1} m_i 2^i$ ,  $P$ )
2:    $(x_0, x_1) \leftarrow (\mathbf{x}(0), \mathbf{x}(P))$ 
3:   for  $i := \beta - 1$  down to 0 do ▷ Montgomery ladder
4:      $(x_{m_i}, x_{-m_i}) \leftarrow (\mathbf{xDBL}(x_{m_i}), \mathbf{xADD}(x_0, x_1, \mathbf{x}(P)))$ 
5:   end for ▷ invariant:  $x_0 = \mathbf{x}(\lfloor m/2^i \rfloor P)$ ,  $x_1 = \mathbf{x}(\lfloor m/2^i \rfloor + 1)P$ 
6:    $Q \leftarrow \text{Recover}(P, x_0, x_1)$  ▷  $Q = [m]P$ 
7:   return  $Q$ 
8: end function
```

---

Now: your fast Kummer implementations can be easily bootstrapped to full Jacobian group implementations; fast Diffie–Hellman code now yields efficient signatures.

# $\mu$ Kummer

## Kummer crypto for microcontrollers *Renes–Schwabe–S.–Batina, CHES 2016*

*A fiendishly simple plan:*

1. Build efficient Diffie–Hellman software for small devices, based on Kummer arithmetic; **Done**
2. Bootstrap it to a full Schnorr signature scheme, using the previous trick; **Done**
3. Take over the world. (**In progress**)

# *$\mu$ Kummer software*

Free, public domain C/assembly implementation of Diffie–Hellman key exchange and Schnorr signatures targeting the 128-bit security level on

**AVR ATMega** (8-bit architecture)

**ARM Cortex M0** (32-bit architecture)

Download it: <http://www.cs.ru.nl/~jrenes>

## Comparison for 8-bit architecture (AVR ATmega):

Protocol	Object	kCycles	Stack bytes
Diffie–Hellman	Curve25519	13900	494
	$\mu$ Kummer	9513 (68%)	99 (20%)
Schnorr signing	Ed25519	19048	1473
	$\mu$ Kummer	10404 (55%)	926 (63%)
Schnorr verifying	Ed25519	30777	1226
	$\mu$ Kummer	16241 (53%)	992 (75%)

## Comparison for 32-bit architecture (ARM Cortex M0):

Multiplication for	Object	kCycles	Stack bytes
Diffie–Hellman	Curve25519	3590	548
	$\mu$ Kummer	2634 (73%)	248 (45%)
Schnorr	NIST-P256	10730	540
	$\mu$ Kummer	2709 (25%)	968 (179%)

*Curve25519 = Düll-Haase-Hinterwälder-Hutter-Paar-Sánchez-Schwabe '15*

*Ed25519 = Nascimento-López-Dahab '15*

*NIST-P256 = Wenger-Unterluggauer-Werner '13*

Recovery:

<http://eprint.iacr.org/2016/777>

$\mu$ Kummer:

<http://eprint.iacr.org/2016/366>

Download the software:

<http://www.cs.ru.nl/~jrenes>