Geometry meets IoT: Efficient low-memory key exchange and signatures

Benjamin Smith

Team **GRACE**

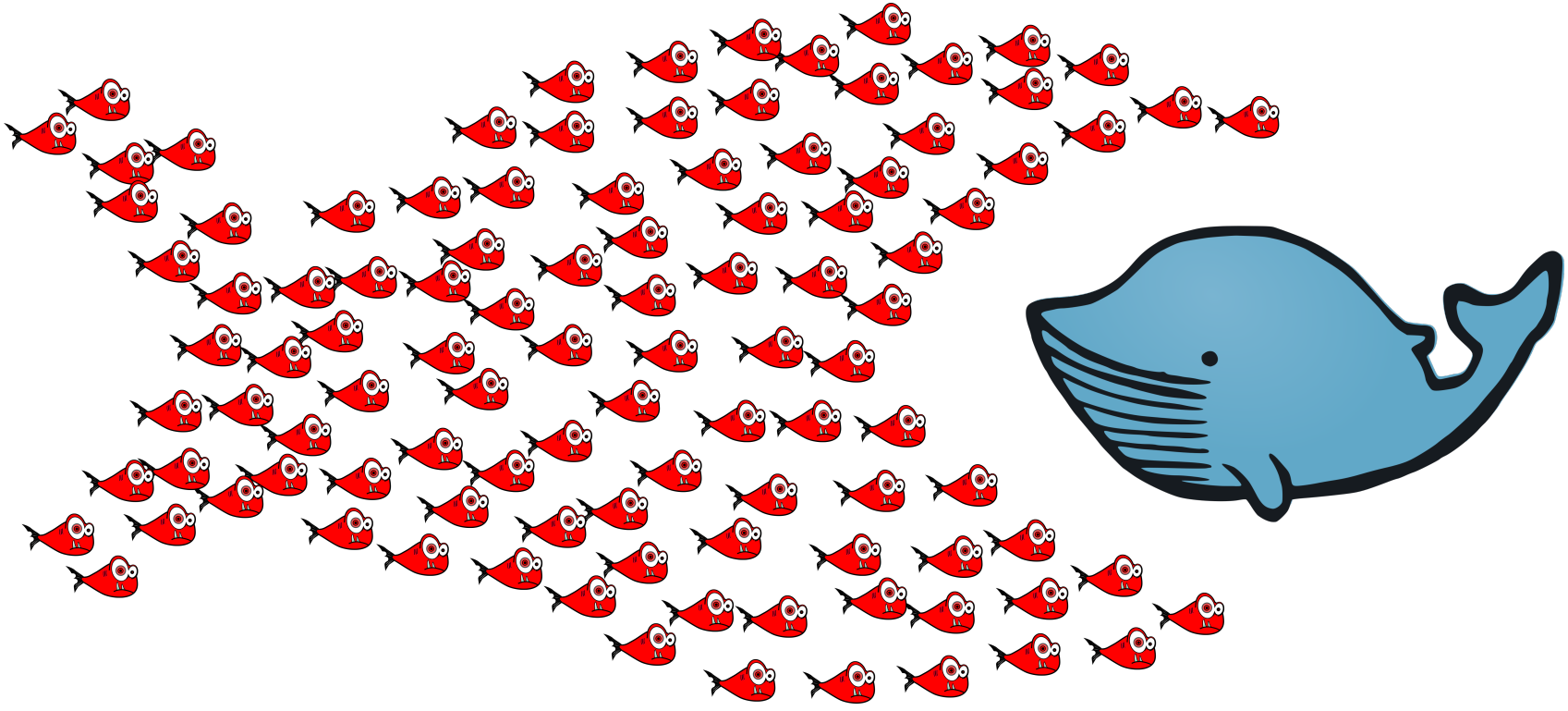
INRIA + Laboratoire d'Informatique de l'École polytechnique (LIX)

Summer school on real-world crypto and privacy

Sibenik, Croatia, June 8 2017

1. The problem space

IoT = a ubiquitous, pervasive, embedded, decentralised distributed computing platform.



Virtually **unsecured**, and mostly **unmaintained**.
Society is totally exposed and vulnerable.

We want to secure IoT with a mixture of symmetric and asymmetric crypto (as in Bart Preneel's talk).

Unfortunately, embarking asymmetric crypto on a microcontroller is like "carrying a sofa on a motorbike".

Think about implementing, say, RSA signatures: apart from a bit of (easy) hashing, you just need to cube a 384-byte integer modulo another 384-byte integer.

Easy!

If you only have, say, 1K of RAM, then this kind of thing is impossible.

When it comes to security, there is no “half a sofa”.

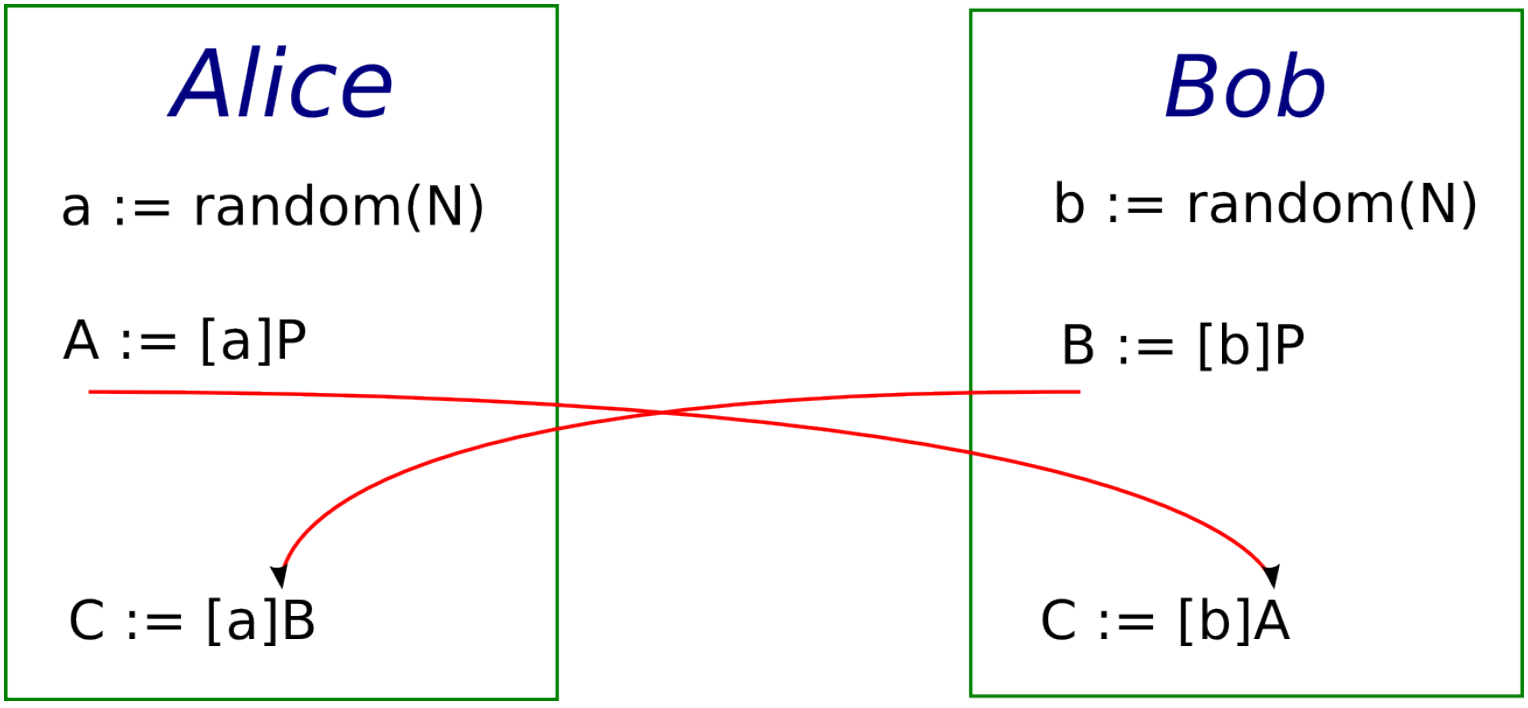
IoT needs full-sized security, simply because our adversaries do not have the same constraints on power, time, memory, access.

This talk: developing more streamlined, aerodynamic sofas.

Also, more efficient public-key crypto algorithms: fast signatures in well under 1K of RAM.

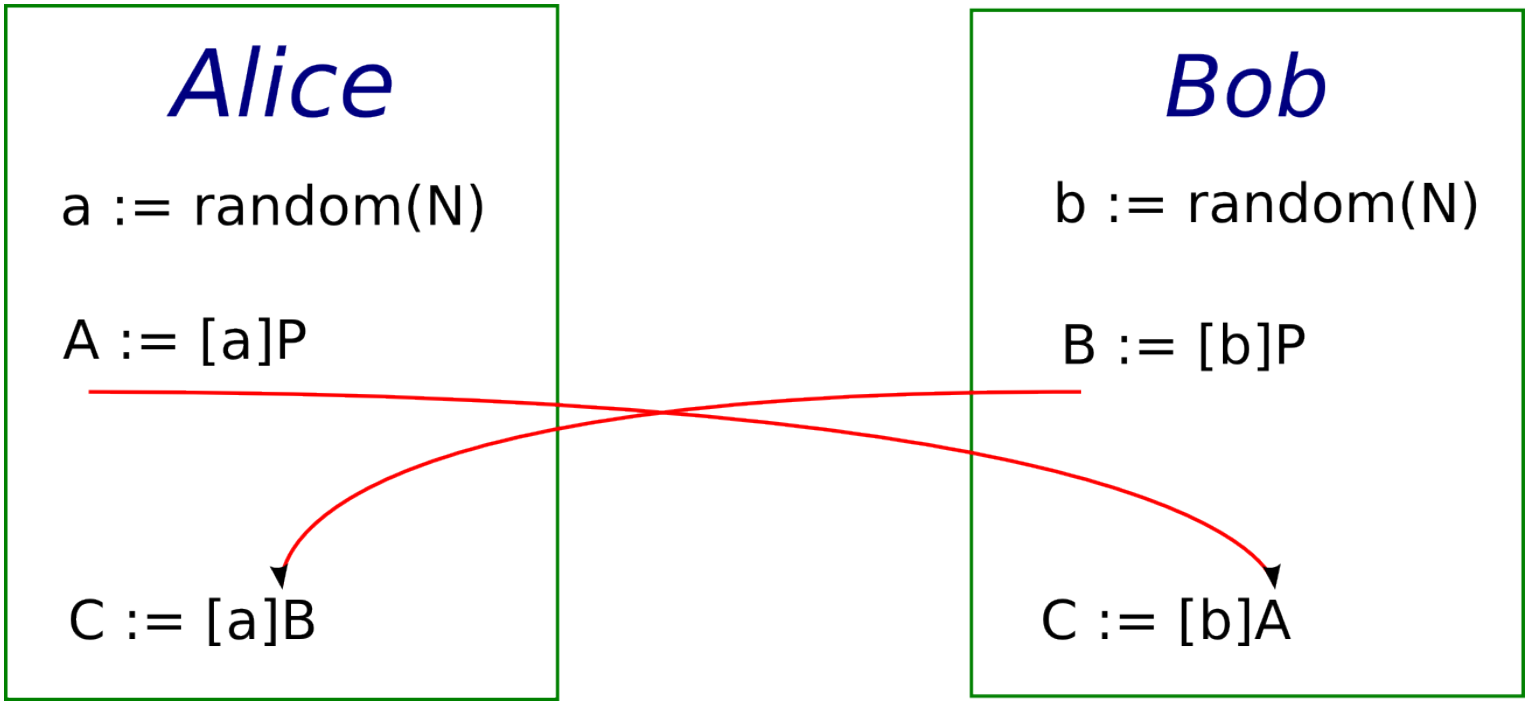
2. Modern Diffie–Hellman: X25519

Diffie–Hellman key exchange: classic view



- $\mathcal{G} = \langle P \rangle$ is a cyclic group
- a, b secret integers
- Security: Computational Diffie–Hellman Problem (CDHP)
  Practical cryptographic groups $\mathcal{G}$: CDHP $\equiv$ Discrete Log

Diffie–Hellman key exchange: modern view



- $\mathcal{G}$ is just a set, not a group!
- $[a]$, $[b]$ secret commuting maps $\mathcal{G} \to \mathcal{G}$.
- CDHP: reduce to CDHP/Discrete Log in groups.

Candidates for Diffie–Hellman systems

1970s/80s *Set $\mathcal{G}$*: subgroup of $\mathbb{G}_m(\mathbb{F}_p)$.

Maps: random exponentiations.

CDHP: in $\mathbb{G}_m(\mathbb{F}_p)$.

90s/2000s *Set $\mathcal{G}$*: subgroup of an elliptic curve $\mathcal{E}(\mathbb{F}_p)$

Maps: random scalar multiplications on $\mathcal{E}$.

CDHP: in $\mathcal{E}(\mathbb{F}_p)$.

Advantage: MUCH smaller $p \implies$ fast, compact.

2006→ *Set $\mathcal{G}$:* $(\mathcal{E}/\pm 1)(\mathbb{F}_p) = \mathbb{P}^1(\mathbb{F}_p) = \log_2 q$*-bit strings*

Maps: random commuting $\mathbb{P}^1 \to \mathbb{P}^1$ (from $\mathcal{E}$).

CDHP: in $\mathcal{E}(\mathbb{F}_p)$ & quadratic twist.

Advantage: faster, more compact, fault-tolerant.

Moving from $\mathcal{E}$ to $\mathbb{P}^1 = \mathcal{E}/\pm 1$

$$(X, Y, Z) \in \mathcal{E} \longmapsto (X : Z) \in \mathbb{P}^1 = \mathcal{E}/\pm 1\,.$$

The group law $+$ on $\mathcal{E}$ is lost on $\mathbb{P}^1$, **but** we retain well-defined "scalar multiplications"

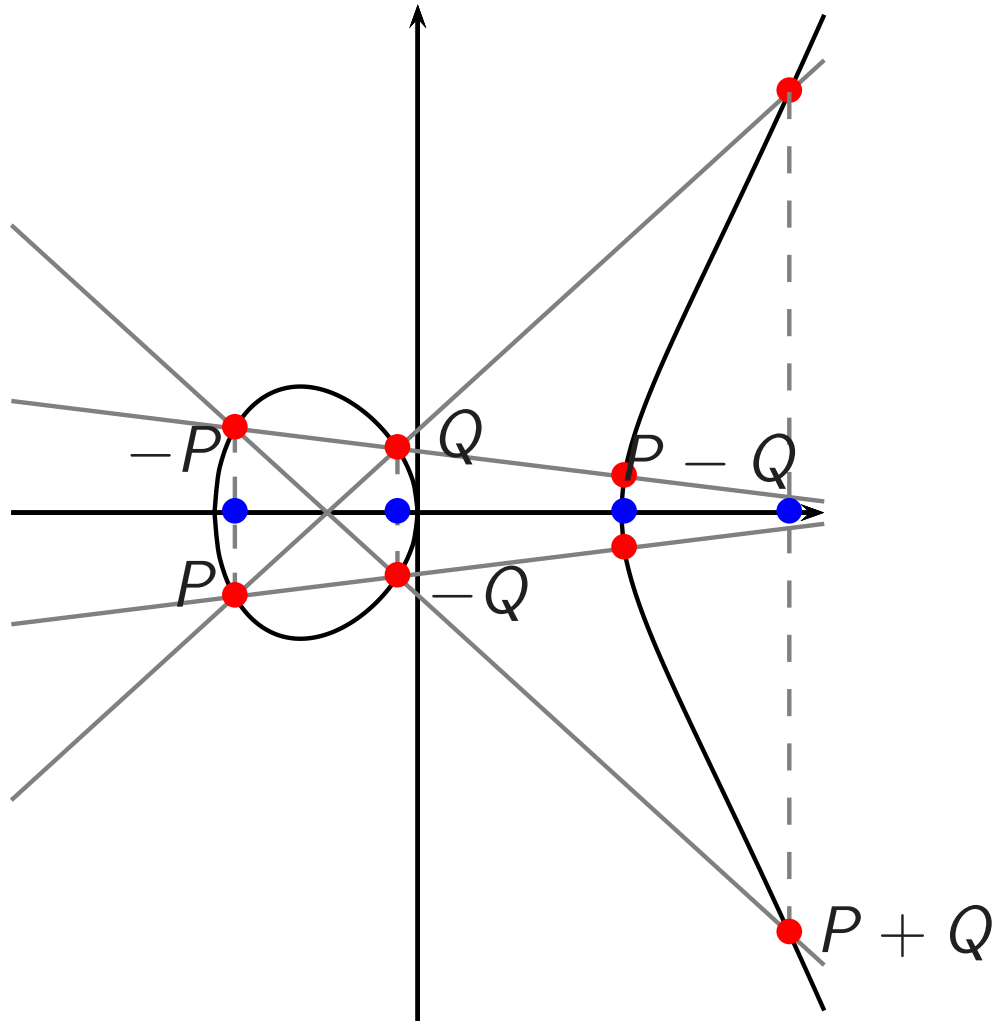
$$[m] : \pm P \longmapsto \pm [m]P$$

because $-[m](P) = [m](-P)$.

Problem: Compute $[m]$ efficiently *without* $+$.

Observe:
$\{\pm P, \pm Q\}$ determines $\{\pm(P+Q), \pm(P-Q)\}$.

$\{\pm P, \pm Q)\}$ *determines* $\{\pm(P-Q), \pm(P+Q)\}$



Any 3 of $\pm P$, $\pm Q$, $\pm(P-Q)$, $\pm(P+Q)$ determines the 4th

Since any 3 of $\pm P$, $\pm Q$, $\pm(P-Q)$, $\pm(P+Q)$ determines the 4th, we can define

pseudo-addition

$$\texttt{xADD} : (\pm P, \pm Q, \pm(P-Q)) \longmapsto \pm(P+Q)$$

pseudo-doubling

$$\texttt{xDBL} : \pm P \longmapsto \pm[2]P$$

$\Longrightarrow$ Evaluate $[m]$ by combining `xADD`s and `xDBL`s using differential addition chains

(ie. every $+$ has summands with known difference)

Example: the classic *Montgomery ladder*.

The Montgomery ladder

Algorithm 1: The Montgomery ladder

Input: $m = \sum_{i=0}^{\beta-1} m_i 2^i$, P
Output: $[m]P$

1 $(R_0, R_1) \leftarrow (\mathcal{O}_{\mathcal{E}}, P)$
2 **for** $i := \beta - 1$ *down to* 0 **do**
  // invariant: $(R_0, R_1) = ([\lfloor m/2^i \rfloor]P, [\lfloor m/2^i \rfloor + 1]P)$
3 **if** $m_i = 0$ **then**
4 $(R_0, R_1) \leftarrow ([2]R_0, R_0 + R_1)$
5 **else**
6 $(R_1, R_0) \leftarrow ([2]R_1, R_0 + R_1)$
7 **return** R_0 // $R_0 = [m]P$, $R_1 = [m]P + P$

We replace the **if** statement branch with a constant-time conditional swap; then the whole ladder becomes uniform and constant-time, which is important for side-channel protection.

The x-only Montgomery ladder

Algorithm 2: The Montgomery ladder

Input: $m = \sum_{i=0}^{\beta-1} m_i 2^i$, P
Output: $[m]P$

1 $(R_0, R_1) \leftarrow (\pm 0, \pm P)$
2 **for** $i := \beta - 1$ *down to* 0 **do**
  // inv.: $(R_0, R_1) = (\pm[\lfloor m/2^i \rfloor]P, \pm[\lfloor m/2^i \rfloor + 1]P)$
3 **if** $m_i = 0$ **then**
4 $(R_0, R_1) \leftarrow (\mathtt{xDBL}(R_0), \mathtt{xADD}(R_0, R_1, \pm P))$
5 **else**
6 $(R_1, R_0) \leftarrow (\mathtt{xDBL}(R_1), \mathtt{xADD}(R_0, R_1, \pm P))$
7 **return** R_0 // $R_0 = \pm[m]P$, $R_1 = \pm([m+1]P)$

Note: xDBL and xADD share some operands.
$\Longrightarrow$ combine them in a faster xDBLADD operation.

Montgomery models of elliptic curves

$$\mathcal{E} : \Delta Y^2 Z = X(X^2 + cXZ + Z^2)$$

with curve constant c and "twisting constant" Δ in $\mathbb{F}_p$.

The map $x : \mathcal{E} \to \mathbb{P}^1$ is $x : (X : Y : Z) \longmapsto (X : Z)$.

- $\texttt{xADD}((X_P : Z_P), (X_Q : Z_Q), (X_{P-Q} : Z_{P-Q}))$
  $= (Z_{P-Q}(S_P T_Q + T_P S_Q)^2 : X_{P-Q}(S_P T_Q - T_P S_Q)^2)$
  where $S_P := X_P - Z_P$, $T_P := X_P + Z_P$, etc.
- $\texttt{xDBL}((X : Z)) = (UV : W(U + \frac{c+2}{4} W))$
  where $U = (X + Z)^2$, $V = (X - Z)^2$, $W = U - V$.

Observe that Δ *never appears* in these operations!

What is the elliptic curve doing?

Diffie–Hellman is now defined by "secret functions" $[a]$ and $[b]$, each of which is a series of $\log_2 q$ random `CSwaps` followed by

$$(T_0, T_1) \longmapsto (\texttt{xDBL}(T_0), \texttt{xADD}(T_0, T_1, X)).$$

where $X =$ the public generator $\pm P$
or a public key $\pm A$ (or $\pm B$), depending on the protocol step.

One system parameter, $c \in \mathbb{F}_p \longleftrightarrow$ curve $\mathcal{E}$, which

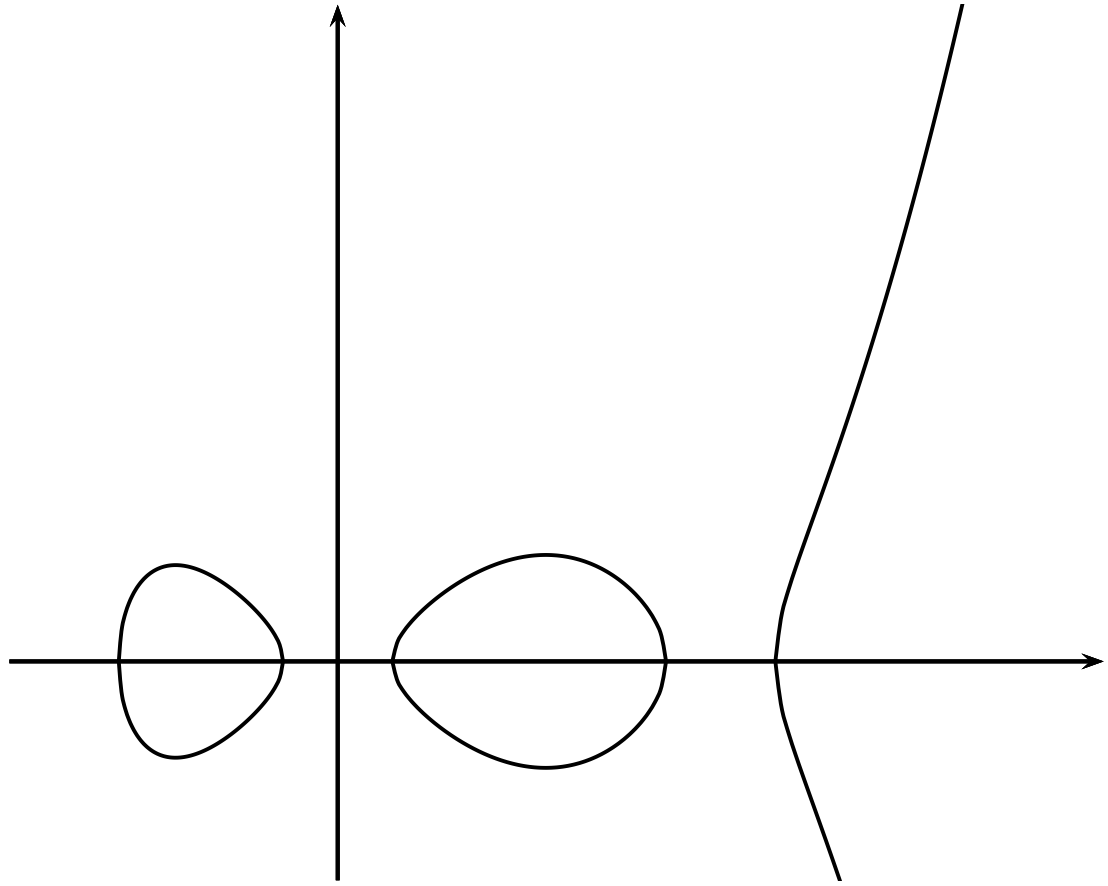
- Defines the operation `xDBL` (`xADD` *is independent of* $\mathcal{E}$)
- Proves that the secret functions $[a], [b]$ commute
- Gives hard upper and conjectural lower bounds on security (from the CDHP on on $\mathcal{E}$ and its quadratic twist)

If we take $c = 486662$ and $p = 2^{255} - 19$, then $\mathcal{E}$ is Bernstein's **Curve25519**, and the key exchange is known as **X25519**.

3. Faster Diffie–Hellman with Kummer surfaces

Genus 2 curves

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



Unlike elliptic curves, the points do not form a group.

Making groups from genus 2 curves

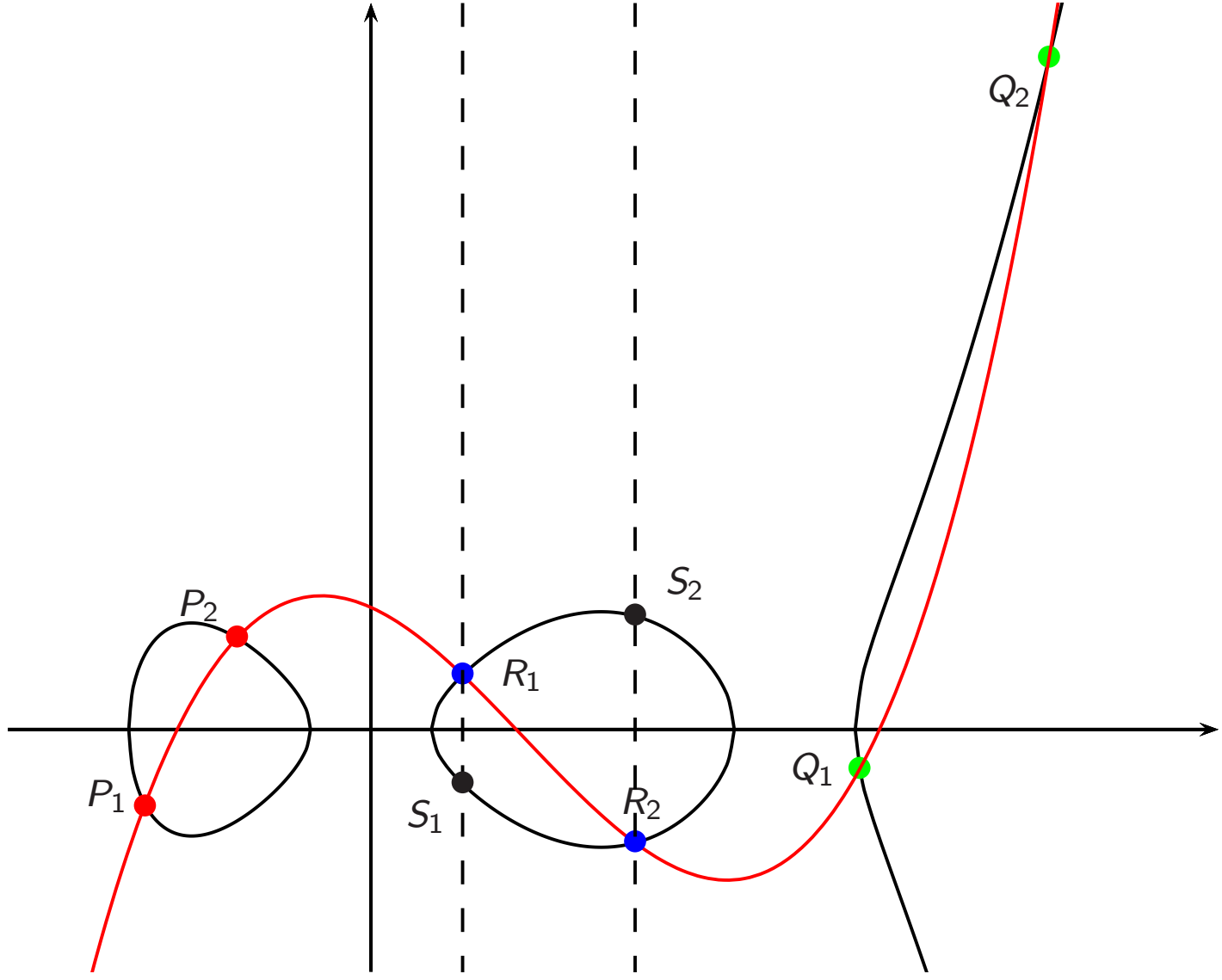
Jacobian: algebraic group $\mathcal{J}_{\mathcal{C}} \cong \mathrm{Pic}^0(\mathcal{C})$;
geometrically, $\mathcal{J}_{\mathcal{C}} \sim \mathcal{C}^{(2)}$ (symmetric square of $\mathcal{C}$)

(with all pairs $\{(x, y), (x, -y)\}$ "blown down" to 0)

Group law on $\mathcal{J}_{\mathcal{C}}$ induced by
$\{P_1, P_2\} + \{Q_1, Q_2\} + \{R_1, R_2\} = 0$
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 plane points determine a cubic
$y = g(x)$; and $y = g(x)$ intersects $\mathcal{C} : y^2 = f(x)$
in 6 places because $g(x)^2 = f(x)$ has 6 solutions.

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



What is the Jacobian?

$\mathcal{J}_\mathcal{C} \sim \mathcal{C}^{(2)} \implies \mathcal{J}_\mathcal{C}$ is a surface.

Points in $\mathcal{J}_\mathcal{C}(\mathbb{F}_p) \longleftrightarrow$ pairs $\{P_1, P_2\}$ of points of $\mathcal{C}$ with P_1, P_2 both in $\mathcal{C}(\mathbb{F}_p)$ or conjugate in $\mathcal{C}(\mathbb{F}_{p^2})$

$$\implies \#\mathcal{J}_\mathcal{C}(\mathbb{F}_p) = O(p^2).$$

More precisely: $(\sqrt{p}-1)^{2\times 2} \le \#\mathcal{J}_\mathcal{C}(\mathbb{F}_p) \le (\sqrt{p}+1)^{2\times 2}$.

Replace 2s with 1s $\longrightarrow$ elliptic curves (genus 1).

Abstractly: $\mathcal{J}_\mathcal{C}(\mathbb{F}_p)$ drop-in replacement for some $\mathcal{E}(\mathbb{F}_q)$ (but only need $\log_2 p \approx \frac{1}{2}\log_2 q$).

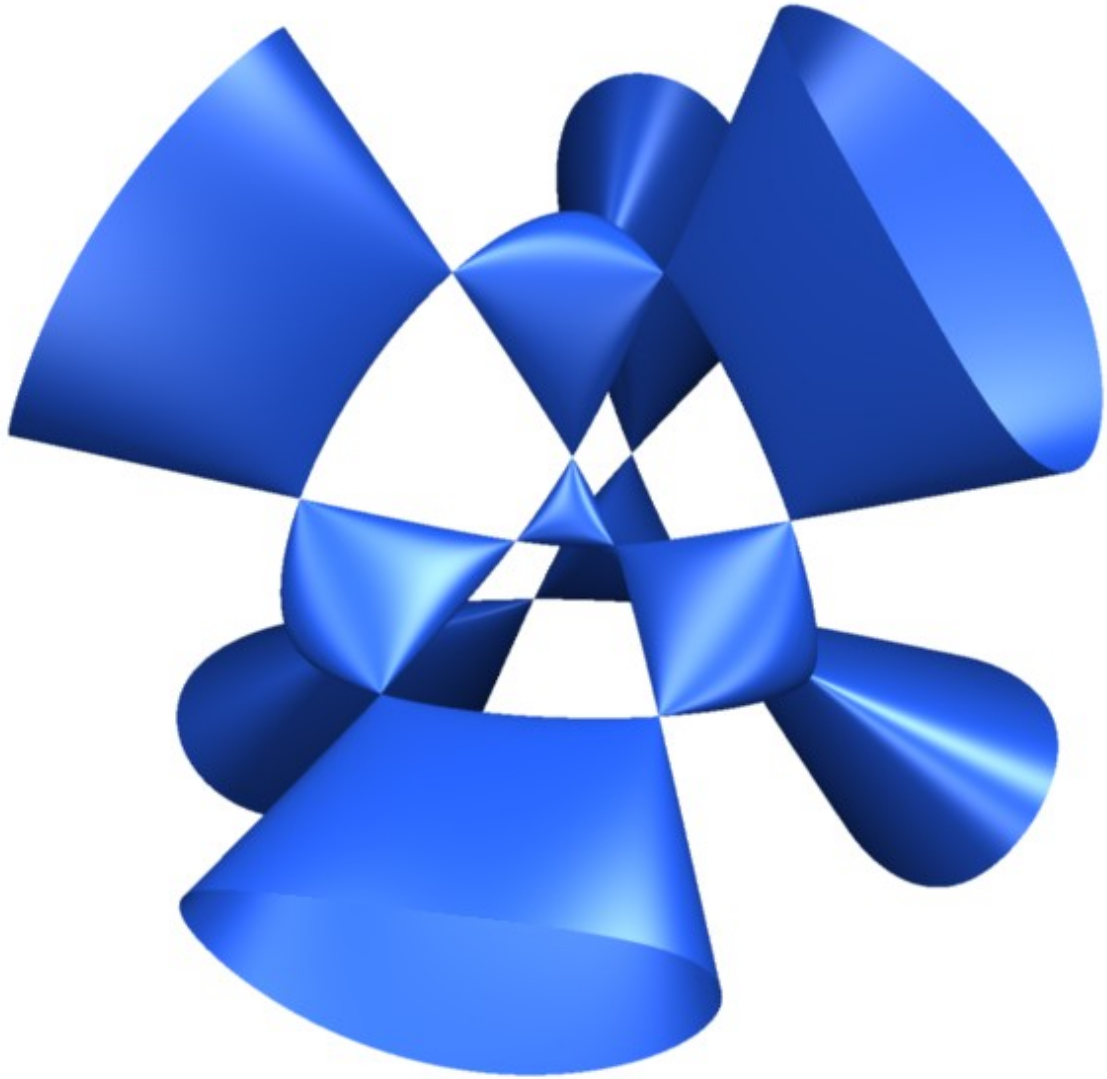
But the algorithms and geometry of $\mathcal{J}_\mathcal{C}$ are *much* more complicated than for $\mathcal{E}$.

Kummer varieties

If $\mathcal{E} : y^2 = f(x)$ is an elliptic curve,
then $-(x, y) = (x, -y)$;
so $P \mapsto x(P)$ is the quotient by ± 1.
$\Longrightarrow$ the x-line $\mathbb{P}^1$ is the *Kummer variety* of $\mathcal{E}$.

Genus 2 analogue of the x-line $\mathbb{P}^1$:
The *Kummer surface* $\mathcal{K}_\mathcal{C} := \mathcal{J}_\mathcal{C}/\langle \pm 1 \rangle$
is a quartic surface in $\mathbb{P}^3$ with 16 point singularities
(which are the images of the 16 points in $\mathcal{J}_\mathcal{C}[2]$*).*

What a Kummer surface looks like



...This is the genus 2 analogue of what is just a line for elliptic curves, which says a lot about the jump in mathematical complexity...

Kummer surfaces

The classical model of the Kummer surface for $\mathcal{C}$:

$$X^4 + Y^4 + Z^4 + W^4 + 2E \cdot XYZW$$
$$= F(X^2W^2 + Y^2Z^2) + G(X^2Z^2 + Y^2W^2) + H(X^2Y^2 + Z^2W^2)$$

where E, F, G, H are constants related to $\mathcal{C}$.

$\mathcal{K}_{\mathcal{C}}$ is not a group, but we get scalar multiplication from $\mathcal{J}_{\mathcal{C}}$ *(since* $[m](-D) = -[m]D$*)*.

Faster than elliptic x-line arithmetic at the same security level (Chudnovsky & Chudnovsky, Gaudry, . . .)

Eg. 128-bit security: $\mathcal{K}_{\mathcal{C}}$ over 128-bit field beats $\mathcal{E}$ over 256-bit field

Kummer surface arithmetic

We define

$$\begin{aligned}\mathcal{M} : ((x_1 : y_1 : z_1 : t_1), (x_2 : y_2 : z_2 : t_2)) &\longmapsto (x_1x_2 : y_1y_2 : z_1z_2 : t_1t_2)\,,\\ \mathcal{S} : (x : y : z : t) &\longmapsto (x^2 : y^2 : z^2 : t^2)\,,\\ \mathcal{I} : (x : y : z : t) &\longmapsto (1/x : 1/y : 1/z : 1/t)\end{aligned}$$

and the *Hadamard transformation*

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

Then we can use these operations for the Montgomery ladder:

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

(The green things here are essentially constants)

Kummer surfaces in practice

Kummers are already used for high-speed Diffie–Hellman

E.g.: Bos–Costello–Hisil–Lauter, 2012;
Bernstein–Chuengsatiansup–Lange–Schwabe, 2014

Moving to microcontrollers, μKummer
(Renes–Schwabe–S.–Batina, CHES 2016):
Open crypto lib for 8- and 32-bit microcontrollers.

	AVR ATmega (8-bit)		ARM Cortex M0 (32-bit)	
	KCycles	Stack bytes	KCycles	Stack bytes
NIST P-256	34930	590	10730	540
Curve25519	13900	494	3590	548
μKummer	**9739**	**99**	**2644**	**248**

NIST P-256 = Wenger–Unterluggauer–Werner (2013)
Curve25519 = Düll–Haase–Hinterwälder–Hutter–Paar–Sánchez–Schwabe (2015)

Kummer point compression

Problem: traditionally (since 2006),
public key Kummer points $\pm Q = (X_Q : Y_Q : Z_Q : T_Q)$ are transmitted as $(u, v, w) = (X_Q/Y_Q, X_Q/Z_Q, X_Q/T_Q) \in \mathbb{F}_p^3$.

Convenient for arithmetic: we need $\mathcal{I}(\pm Q) = (1 : u : v : w)$ at the start of the ladder anyway, but it meant that
Kummer DH keys were 50% larger than elliptic DH keys
(eg. 3×128 versus 1×256 bits).

Mathematically we should compress to $2 \times \log_2 p + \epsilon$ bits (because $\mathcal{K}_\mathcal{C}$ is a surface), but this looked algorithmically painful because the defining equation of $\mathcal{K}_\mathcal{C}$ is quartic.

New Kummer compression scheme

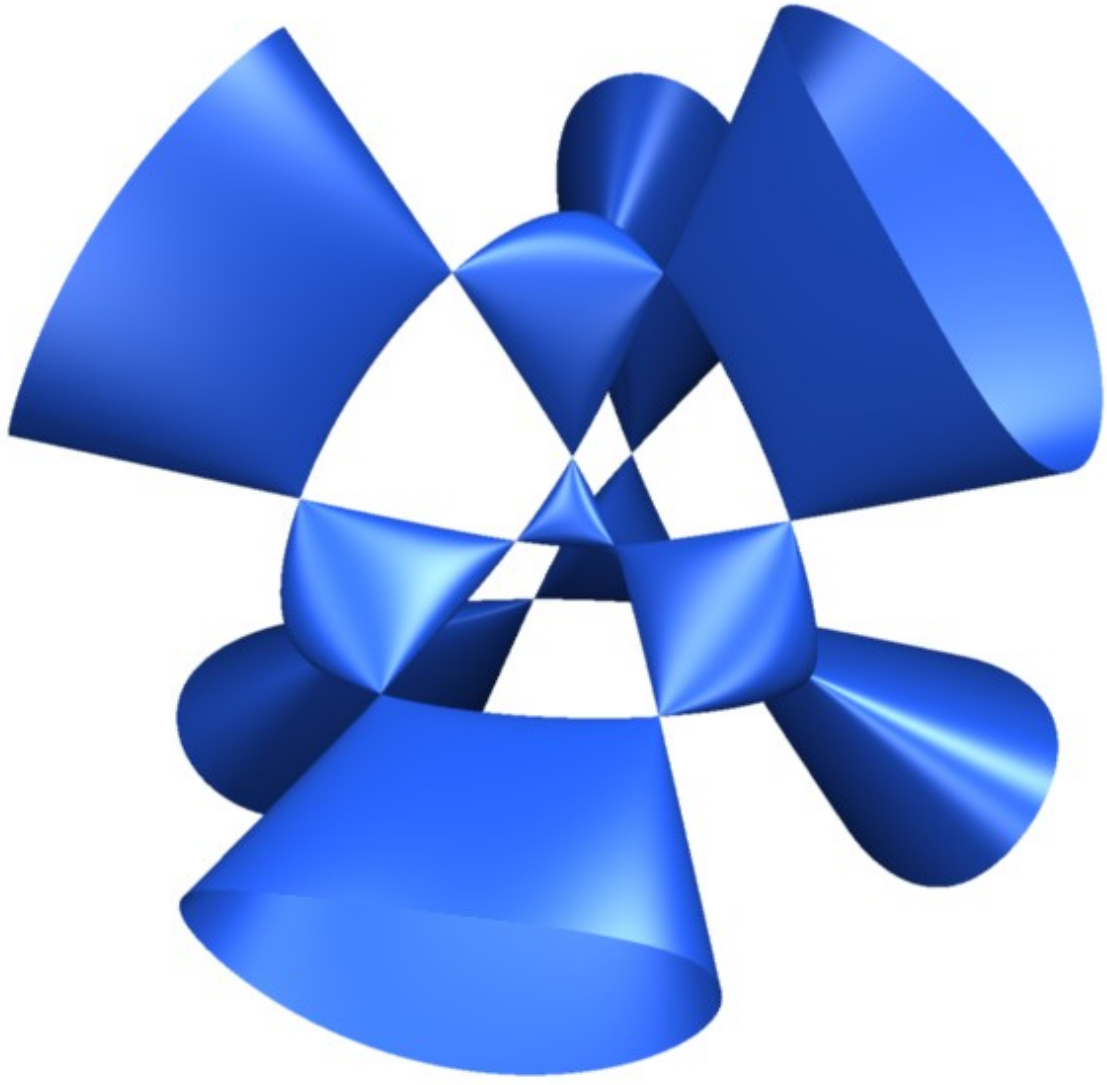
New solution (Renes–S. 2017):
use geometry for efficient Kummer compression.

If we map the coordinates of four special nodes
(the kernel of an isogeny splitting xDBL)
to the corners of a coordinate tetrahedron in $\mathbb{P}^3$, then
the defining equation becomes
a sparse quadratic in *all four* variables!

We can recover the value of any coordinate from the three
others plus a single square root (controlled by one bit).

Normalizing the other 3 coordinates,
we compress Kummer points to $2 \times \log_2 p + 2$ bits.

Exercise: visualise the compression



4. Signatures for microcontrollers

Signatures for microcontrollers

Kummer surfaces are a good solution for compact, fast Diffie–Hellman.

Problem: we also want signatures, and verifying signatures means checking equations like

$$R = [s]P + [e]Q$$

where R, P, Q are in a group.
Kummer surfaces have no group law $+$...

How can we expoit the speed of Kummer/Montgomery arithmetic for signatures?

Conventional approach: Don't do it.

Don't do it. Use Kummer/Montgomery for Diffie–Hellman, and a separate twisted Edwards curve for signatures. Eg. NaCl library.

Disadvantages:

- slower arithmetic for signatures,
- more stack space for Edwards coordinates,
- two objects $\Longrightarrow$ bigger trusted code base,
- separate public key formats for Diffie–Hellman and signatures.

Hybrid approach: Recovery

Use $\mathbb{P}^1$/Kummer for Diffie–Hellman. For signatures,

1. Start with group elements P;
2. Project P to $\pm P$ on $\mathbb{P}^1$ or the Kummer, and compute scalar multiples there with the ladder;
3. The ladder actually computes $\pm[m]P$ **and** $\pm[m+1]P$, and the triple $(P, \pm[m]P, \pm[m+1]P)$ determines $[m]P$;
4. Use point recovery formulæ to get the correct $[m]P$ back in the curve/Jacobian, and apply the full group law there for signature verification.

Advantages: Kummer speed for signatures.
Disadvantages: still need to implement the group law (bigger trusted code base); still have mixed public key formats; recovery formulæ require a lot of stack space to compute (v. important in the IoT setting).

Putting the hybrid approach into practice

μKummer (Renes–Schwabe–S.–Batina, CHES 2016):
Open crypto lib for 8- and 32-bit microcontrollers.
Efficient Diffie–Hellman *and* Schnorr signatures
using Kummer surfaces and genus-2 point recovery.

	ATmega (8-bit)		Cortex M0 (32-bit)	
	KCycles	Stack bytes	KCycles	Stack
DH	9739	429	2644	584
Keygen	10206	812	2774	1056
Sign	10404	926	2865	1360
Verify	16241	992	4454	1432

Substantially faster and smaller than the elliptic SOA,
but inconveniently large stack requirements.

5. A new approach: qDSA

Signature verification

All this group stuff—twisted Edwards, Jacobians, memory-intensive point recovery—is only required because the signature verification equation

$$R = [s]P + [e]Q \; .$$

requires a $+$, hence a group.
Brutal solution: instead, verify the slightly weaker

$$\pm R = \pm[s]P \pm [e]Q \, .$$

Hamburg's elliptic Strobe library already (informally) does this!

q*uotient* **D***igital* **S***ignature* **A***lgorithm*

Renes–S. 2017: qDSA is a variant of EdDSA (Schnorr-like) using *only* $\mathbb{P}^1$/Kummer arithmetic. A very cheap extension of Diffie–Hellman systems to provide signature schemes.

1. Key pairs: $(\pm Q, x)$ such that $\pm Q = \pm[x]P$.
   Eg. $\pm Q$ is a (compressed) Kummer point, or a Curve25519 key.
2. Signatures are $(\pm R, s)$ such that
   $\pm R \in \{\pm([s]P + [e]Q), \pm([s]P - [e]Q)\}$.

Advantages: unified public-key formats, only fast Montgomery/Kummer arithmetic. And, it turns out, lower stack space requirements!

Checking $\pm R \in \{\pm([s]P \pm [e]Q)\}$

$\{\pm A, \pm B\}$ determines $\{\pm(A+B), \pm(A-B)\}$ for all $\pm A$, $\pm B$;
we need to check if $\pm R \in \{\pm(A+B), \pm(A-B)\}$
where $\pm A = \pm[s]P$ and $\pm B = \pm[e]Q$.

Classical theory of theta functions: there exists
a system of biquadratic homogeneous polynomial
equations in the coordinates of $\pm A$, $\pm B$
that are only satisified by the coordinates of $\pm(A \pm B)$.

Elliptic case on $\mathcal{E} : Y^2Z = X(X^2 + cXZ + Z^2)$:
$\pm R \in \{\pm(A+B), \pm(A-B)\}$ if and only if

$$2B_{XZ} \cdot X_R Z_R = B_{ZZ} \cdot X_R^2 + B_{XX} \cdot Z_R^2 \quad \text{where}$$

$$\begin{aligned}
B_{XX} &= (X_A X_B - Z_A Z_B)^2 , \\
B_{XZ} &= (X_A X_B + Z_A Z_B)(X_A Z_B + Z_A X_B) + 2cX_A Z_A X_B Z_B , \\
B_{ZZ} &= (X_A Z_B - Z_A X_B)^2 .
\end{aligned}$$

Checking $\pm R$ on Kummer surfaces

For Kummer surfaces there are 10 biquadratic forms to evaluate and 6 verification equations to test; this isn't so bad if the forms are in a nice shape.

Unfortunately, the biquadratic forms we need are heavy and dense on $\mathcal{K}_{\mathcal{C}}$...

...But if we break down xDBL into simpler maps, we find that the Hadamard transform takes us into a new isomorphic form of $\mathcal{K}_{\mathcal{C}}$ where the forms are extremely simple to evaluate.

Results

		ATmega (8-bit)		Cortex M0 (32-bit)	
System	Function	Cycles	Stack	Cycles	Stack
Ed25519	sign	19048	1473	—	—
	verify	30777	1226	—	—
Four$\mathbb{Q}$	sign	5175	1590	—	—
	verify	11468	5050	—	—
qDSA-$\mathcal{E}$	sign	14070	412	3889	660
	verify	25375	644	6799	788
μKummer	sign	10404	926	28635	1360
	verify	16240	992	4454	1432
qDSA-$\mathcal{K}_\mathcal{C}$	sign	10477	417	2908	580
	verify	20423	609	5694	808

Ed25519 = Nascimento–López–Dahab (2015)
Four$\mathbb{Q}$ = Liu–Longa–Pereira–Reparaz–Seo (2017)
qDSA-$\mathcal{E}$ = qDSA built over Curve25519
qDSA-$\mathcal{K}_\mathcal{C}$ = qDSA built over the Gaudry–Schost Kummer