μKummer: efficient hyperelliptic signatures and key exchange on microcontrollers

Joost Renes[1] Peter Schwabe[1] Benjamin Smith[2]
Lejla Batina[1]

[1]Digital Security Group, Radboud University, The Netherlands

[2]INRIA *and* Laboratoire d'Informatique de l'École polytechnique (LIX), France

18th August 2016

Outline

- Introduction
- High level signature and key exchange schemes
- Jacobian and Kummer arithmetic
- Implementation details
- Results and comparison

Summary of contributions

1. First software-only implementation of hyperelliptic-curve cryptography on microcontrollers (AVR ATmega and ARM Cortex M0)
2. First implementation of a signature scheme based on a Kummer surface
3. Significant improvement over state-of-the-art in terms of speed, size and stack usage

Software in the public domain. Available at

`http://www.cs.ru.nl/~jrenes/`

Curve-based cryptography

Genus	$g = 1$	$g = 2$
Curve	Elliptic curve E	Hyperelliptic curve $\mathcal{E}$
Cryptographic group	Points	Jacobian
Kummer	$E/\{\pm 1\}$	$\mathcal{K} := \mathcal{J}/\{\pm 1\}$

Curve-based cryptography

Genus	$g = 1$	$g = 2$
Curve	Elliptic curve E	Hyperelliptic curve $\mathcal{E}$
Cryptographic group	Points	Jacobian
Kummer	$E / \{\pm 1\}$	$\mathcal{K} := \mathcal{J} / \{\pm 1\}$

Genus	$g = 1$	$g = 2$
Curve	Elliptic curve E	Hyperelliptic curve $\mathcal{E}$
Cryptographic group	Points	Jacobian
Kummer	$E / \{\pm 1\}$	$\mathcal{K} := \mathcal{J} / \{\pm 1\}$

- Operations

$$\begin{aligned} \texttt{DBL}: &\quad P \mapsto [2]P \\ \texttt{ADD}: &\quad P, Q \mapsto P + Q \end{aligned}$$

- Two main use cases:
  - Key exchange: relies on scalar multiplication $k, P \to [k]P$
  - Signatures: relies on scalar multiplication and addition
- Operations on $\mathcal{J}$ are hard to make fast and constant-time!

Genus	$g = 1$	$g = 2$
Curve	Elliptic curve E	Hyperelliptic curve $\mathcal{E}$
Cryptographic group	Points	Jacobian
Kummer	$E/\{\pm 1\}$	$\mathcal{K} := \mathcal{J}/\{\pm 1\}$

- Corresponds to $(x, y) \mapsto x$
- Not a group. Use x-only operations

$$\texttt{xDBL}: \quad x_P \mapsto x_{[2]P}$$
$$\texttt{xADD}: \quad x_P, x_Q, x_{P\pm Q} \mapsto x_{P\mp Q}$$

- Scalar multiplication via the Montgomery ladder (e.g. Curve25519 [Ber06])
- Main use case: key exchange
- No signatures (e.g. Ed25519 [Ber+12])

Genus	$g = 1$	$g = 2$
Curve	Elliptic curve E	Hyperelliptic curve $\mathcal{E}$
Cryptographic group	Points	Jacobian
Kummer	$E/\{\pm 1\}$	$\mathcal{K} := \mathcal{J}/\{\pm 1\}$

- Not a group. Use operations

$$\mathtt{xDBL}: \quad x_P \mapsto x_{[2]P}$$

$$\mathtt{xADD}: \quad x_P, x_Q, x_{P\pm Q} \mapsto x_{P\mp Q}$$

- Scalar multiplication via the Montgomery ladder
- Main use case: key exchange
- No signatures (need Jacobian)

(Hyper)elliptic curve crypto summarized

The situation in short:

- $E \leftrightarrow \mathcal{J}$
  - Key exchange ✔
  - Signatures ✔
- $E / \{\pm 1\} \leftrightarrow \mathcal{K}$
  - Key exchange ✔
  - Signatures ✗

The situation in short:

- $E \leftrightarrow \mathcal{J}$
  - Key exchange ✔
  - Signatures ✔
- $E/\{\pm 1\} \leftrightarrow \mathcal{K}$
  - Key exchange ✔
  - Signatures ✗

New result [CCS16]; use $\mathcal{K}$ to do fast signatures on $\mathcal{J}$:

$$\begin{array}{ccc} \mathcal{J} & \dashrightarrow & \mathcal{J} \\ \downarrow & & \uparrow \\ \mathcal{K} & \longrightarrow & \mathcal{K} \end{array} \qquad \begin{array}{ccc} P & \dashrightarrow & [k]P \\ \downarrow & & \uparrow \\ x_P & \longrightarrow & (x_{[k]P}, x_{[k+1]P}, x_P) \end{array}$$

PpR: “Project-pseudomultiply-Recover”

Implementation results

- On larger platforms speed records are challenged by Kummer surface implementations [CL15; Ber+14]
- Speed records for 128-bit secure key exchange and signatures on microcontrollers held by elliptic-curve-based schemes

Two interesting questions:

- Q: How well do Kummer-based key exchange schemes perform on microcontrollers?
  - *A: Probably well, but never implemented*
- Q: How do Kummer-based signatures schemes perform?
  - *A: Not clear*

The signature scheme

- Public generator $P \in \mathcal{J}$, 512-bit hash function H, 256-bit secret key d, message M
- Three main functions
  - `keygen`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $Q \leftarrow [16d']P$
  - `sign`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $r \leftarrow H(d''||M)$
    3. $R \leftarrow [r]P$
    4. $h \leftarrow H(R||Q||M)$
    5. $s \leftarrow r - 16h_{128}d' \pmod{\#\mathcal{J}/16}$
    6. $\sigma \leftarrow (h_{128}||s)$
  - `verify`:
    1. $T \leftarrow [s]P + [h_{128}]Q$
    2. $g \leftarrow H(T||Q||M)$
    3. $g_{128} \stackrel{?}{=} h_{128}$

The signature scheme

- Public generator $P \in \mathcal{J}$, 512-bit hash function H, 256-bit secret key d, message M
- Three main functions
  - `keygen`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $Q \leftarrow [16d']P$ (!) Elements of $\mathcal{J}$
  - `sign`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $r \leftarrow H(d''||M)$
    3. $R \leftarrow [r]P$ (!) Elements of $\mathcal{J}$
    4. $h \leftarrow H(R||Q||M)$
    5. $s \leftarrow r - 16h_{128}d' \pmod{\#\mathcal{J}/16}$
    6. $\sigma \leftarrow (h_{128}||s)$
  - `verify`:
    1. $T \leftarrow [s]P + [h_{128}]Q$ (!) Elements of $\mathcal{J}$
    2. $g \leftarrow H(T||Q||M)$
    3. $g_{128} \stackrel{?}{=} h_{128}$

The signature scheme

- Public generator $P \in \mathcal{J}$, 512-bit hash function H, 256-bit secret key d, message M
- Three main functions
  - `keygen`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $Q \leftarrow [16d']P$ (!) Scalarmult through $\mathcal{K}$ via PpR
  - `sign`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $r \leftarrow H(d''||M)$
    3. $R \leftarrow [r]P$ (!) Scalarmult through $\mathcal{K}$ via PpR
    4. $h \leftarrow H(R||Q||M)$
    5. $s \leftarrow r - 16h_{128}d' \pmod{\#\mathcal{J}/16}$
    6. $\sigma \leftarrow (h_{128}||s)$
  - `verify`:
    1. $T \leftarrow [s]P + [h_{128}]Q$ (!) Scalarmult through $\mathcal{K}$ via PpR
    2. $g \leftarrow H(T||Q||M)$
    3. $g_{128} \stackrel{?}{=} h_{128}$

The signature scheme

- Public generator $P \in \mathcal{J}$, 512-bit hash function H, 256-bit secret key d, message M
- Three main functions
  - `keygen`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $Q \leftarrow [16d']P$
  - `sign`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $r \leftarrow H(d''||M)$
    3. $R \leftarrow [r]P$
    4. $h \leftarrow H(R||Q||M)$
    5. $s \leftarrow r - 16h_{128}d' \pmod{\#\mathcal{J}/16}$
    6. $\sigma \leftarrow (h_{128}||s)$ (!) Compressed to 384 bits by sending h_{128}
  - `verify`:
    1. $T \leftarrow [s]P + [h_{128}]Q$ (!) Half-size scalar multiplication
    2. $g \leftarrow H(T||Q||M)$
    3. $g_{128} \stackrel{?}{=} h_{128}$

- Public generator $P \in \mathcal{J}$, 512-bit hash function H, 256-bit secret key d, message M
- Three main functions
  - `keygen`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $Q \leftarrow [16d']P$ (!) Compression of Q
  - `sign`:
    1. $(d'||d'') \leftarrow H(d)$
    2. $r \leftarrow H(d''||M)$
    3. $R \leftarrow [r]P$ (!) Compression of R
    4. $h \leftarrow H(R||Q||M)$
    5. $s \leftarrow r - 16h_{128}d' \pmod{\#\mathcal{J}/16}$
    6. $\sigma \leftarrow (h_{128}||s)$
  - `verify`:
    1. $T \leftarrow [s]P + [h_{128}]Q$ (!) Compression of T
    2. $g \leftarrow H(T||Q||M)$
    3. $g_{128} \stackrel{?}{=} h_{128}$

The key exchange scheme

- Public generator $P \in \mathcal{K}$, 256-bit secret key d
- One main function
  - `dh_exchange`:
    1. $Q \leftarrow [d]P$

- Public generator $P \in \mathcal{K}$, 256-bit secret key d
- One main function
  - `dh_exchange`:
    1. $Q \leftarrow [d]P$ (!) Only on $\mathcal{K}$

The key exchange scheme

- Public generator $P \in \mathcal{K}$, 256-bit secret key d
- One main function
  - `dh_exchange`:
    1. $Q \leftarrow [d]P$ (!) Both `keygen` and `exchange`

Building blocks: Jacobian & Kummer

- Finite field $\mathbb{F}_q$ with $q = 2^{127} - 1$
- The Gaudry-Schost curve $\mathcal{C}$ is a genus 2 hyperelliptic curve

$$\mathcal{C} : Y^2 = X(X-1)(X-\lambda)(X-\mu)(X-\nu),$$

  for constants $\lambda, \mu, \nu \in \mathbb{F}_q$
- Jacobian $\mathcal{J}_\mathcal{C}(\mathbb{F}_q)$
- Kummer surface $\mathcal{K}_\mathcal{C}(\mathbb{F}_q) := \mathcal{J}_\mathcal{C}(\mathbb{F}_q)/\{\pm 1\}$

Function	Domain & Range	**M**	**S**	$\mathbf{m_c}$	**a**	**s**	**I**
`ADD`	$\mathcal{J}_\mathcal{C} \to \mathcal{J}_\mathcal{C}$	28	2	0	11	24	0
`Project`	$\mathcal{J}_\mathcal{C} \to \mathcal{K}_\mathcal{C}$	8	1	4	7	8	0
`xDBLADD`	$\mathbb{Z} \times \mathcal{K}_\mathcal{C} \to \mathcal{K}_\mathcal{C}^2$	7	12	12	16	16	0
`Recover`	$\mathcal{J}_\mathcal{C} \times \mathcal{K}_\mathcal{C}^3 \to \mathcal{J}_\mathcal{C}$	77	8	0	19	10	1

Building blocks: finite-field arithmetic

AVR ATmega

- Family of 8-bit microcontrollers
- Represent elements of $\mathbb{F}_{2^{127}-1}$ with 16 8-bit words (1 bit left)
- 128×128-bit multiplication (`bigint_mul`) and squaring (`bigint_sqr`) from [HS15]
  - 2-level Karatsuba multiplication and 1-level Karatsuba squaring
- Reduction (`bigint_red`) based on $2^{128} \equiv 2 \pmod{2^{127} - 1}$
- Combined into field multiplication (`gfe_mul`) and squaring (`gfe_sqr`)
- Fast 16×128-bit multiplication by constant (`gfe_mulconst`)
- Inversion (`gfe_invert`) based on $g^{-1} = g^{2^{127}-3}$

Building blocks: finite-field arithmetic

ARM Cortex M0

- 32-bit microcontroller
- Represent elements of $\mathbb{F}_{2^{127}-1}$ with 4 32-bit words (1 bit left)
- 128×128-bit multiplication (`bigint_mul`) and squaring (`bigint_sqr`) from [Dül+15]
  - 2-level Karatsuba multiplication and 2-level Karatsuba squaring
- Reduction (`bigint_red`) based on $2^{128} \equiv 2 \pmod{2^{127}-1}$
- Combined into field multiplication (`gfe_mul`) and squaring (`gfe_sqr`)
- Fast 16×128-bit multiplication by constant (`gfe_mulconst`)
- Inversion (`gfe_invert`) based on $g^{-1} = g^{2^{127}-3}$

AVR ATmega (scalarmult)

	Imp.	Object	Cycles	Code size	Stack
DH	[LWG14]	256-bit curve	≈ 21 078 200	14 700 bytes	556 bytes
S,DH	[WUW13]	NIST P-256	≈ 34 930 000	16 112 bytes	590 bytes
DH	[HS13]	Curve25519	22 791 579	n/a	677 bytes
DH	[Dül+15]	Curve25519	13 900 397	17 710 bytes	494 bytes
DH	**This work**	$\mathcal{K}_{\mathcal{C}}$	9 513 536	≈ 9 490 bytes	99 bytes
S	**This work**	$\mathcal{J}_{\mathcal{C}}$	9 968 127	≈ 16 516 bytes	735 bytes

Results and comparison

AVR ATmega

	Imp.	Object	Cycles	Code size	Stack
DH	[LWG14]	256-bit curve	$\approx$ 21 078 200	14 700 bytes	556 bytes
S,DH	[WUW13]	NIST P-256	$\approx$ 34 930 000	16 112 bytes	590 bytes
DH	[HS13]	Curve25519	22 791 579	n/a	677 bytes
DH	[Dül+15]	Curve25519	13 900 397	17 710 bytes	494 bytes
DH	**This work**	$\mathcal{K}_\mathcal{C}$	9 513 536	$\approx$ 9 490 bytes	99 bytes
S	**This work**	$\mathcal{J}_\mathcal{C}$	9 968 127	$\approx$ 16 516 bytes	735 bytes

Key exchange: Reducing number of clock cycles by 32%, almost halving code size and reducing stack usage by about 80%

AVR ATmega

	Imp.	Object	Cycles	Code size	Stack
DH	[LWG14]	256-bit curve	≈ 21 078 200	14 700 bytes	556 bytes
S,**DH**	[WUW13]	NIST P-256	≈ 34 930 000	16 112 bytes	590 bytes
DH	[HS13]	Curve25519	22 791 579	n/a	677 bytes
DH	[Dül+15]	Curve25519	13 900 397	17 710 bytes	494 bytes
DH	**This work**	$\mathcal{K}_{\mathcal{C}}$	9 513 536	≈ 9 490 bytes	99 bytes
S	**This work**	$\mathcal{J}_{\mathcal{C}}$	9 968 127	≈ 16 516 bytes	735 bytes

Signatures: Reducing number of clock cycles by 71%, increasing stack usage by 25%

Results and comparison

AVR ATmega (full signatures)

Imp.	**Object**	**Function**	**Cycles**	**Stack**
[NLD15]	Ed25519	sig. gen.	19 047 706	1 473 bytes
[NLD15]	Ed25519	sig. ver.	30 776 942	1 226 bytes
This work	$\mathcal{J}_{\mathcal{C}}$	`sign`	10 404 033	926 bytes
This work	$\mathcal{J}_{\mathcal{C}}$	`verify`	16 240 510	992 bytes

Almost half the number of cycles, decrease stack usage (code size not reported)

ARM Cortex M0 (scalarmult)

	Imp.	Object	Clock cycles	Code size	Stack
S,DH	[WUW13]	NIST P-256	$\approx$ 10 730 000	7 168 bytes	540 bytes
DH	[Dül+15]	Curve25519	3 589 850	7 900 bytes	548 bytes
DH	**This work**	$\mathcal{K}_{\mathcal{C}}$	2 633 662	$\approx$ 4 328 bytes	248 bytes
S	**This work**	$\mathcal{J}_{\mathcal{C}}$	2 709 401	$\approx$ 9 874 bytes	968 bytes

ARM Cortex M0

	Imp.	Object	Clock cycles	Code size	Stack
S,DH	[WUW13]	NIST P-256	$\approx$ 10 730 000	7 168 bytes	540 bytes
DH	[Dül+15]	Curve25519	3 589 850	7 900 bytes	548 bytes
DH	**This work**	$\mathcal{K}_{\mathcal{C}}$	2 633 662	$\approx$ 4 328 bytes	248 bytes
S	**This work**	$\mathcal{J}_{\mathcal{C}}$	2 709 401	$\approx$ 9 874 bytes	968 bytes

Key exchange: Reducing number of clock cycles by 27%, halving code size and stack usage

ARM Cortex M0

	Imp.	Object	Clock cycles	Code size	Stack
S,**DH**	[WUW13]	NIST P-256	≈ 10 730 000	7 168 bytes	540 bytes
DH	[Dül+15]	Curve25519	3 589 850	7 900 bytes	548 bytes
DH	**This work**	$\mathcal{K}_{\mathcal{C}}$	2 633 662	≈ 4 328 bytes	248 bytes
S	**This work**	$\mathcal{J}_{\mathcal{C}}$	2 709 401	≈ 9 874 bytes	968 bytes

Signatures: Reducing number of clock cycles by 75%, increase in code size and stack usage

Thanks for your attention!

[Ber+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe and Bo-Yin Yang. "High-speed high-security signatures". In: *J. Cryptographic Engineering* 2.2 (2012). https://cryptojedi.org/papers/#ed25519, pp. 77–89.

[Ber+14] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange and Peter Schwabe. "Kummer Strikes Back: New DH Speed Records". In: *Advances in Cryptology – ASIACRYPT 2014*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. https://cryptojedi.org/papers/#kummer. Springer, 2014, pp. 317–337.

[Ber06] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias and Tal Malkin. Vol. 3958. Lecture Notes in Computer Science. Springer, 2006, pp. 207–228. DOI: 10.1007/11745853_14. URL: http://dx.doi.org/10.1007/11745853_14.

References II

[CCS16] Ping-Ngai Chung, Craig Costello and Benjamin Smith. "Fast, Uniform Scalar Multiplication for Genus 2 Jacobians with Fast Kummers". In: *Selected Areas in Cryptography - SAC 2016, 23rd Conference on Selected Areas in Cryptography.* https://eprint.iacr.org/2015/983. 2016.

[CL15] Craig Costello and Patrick Longa. "Four$\mathbb{Q}$: Four-Dimensional Decompositions on a $\mathbb{Q}$-curve over the Mersenne Prime". In: *Advances in Cryptology – ASIACRYPT 2015*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9452. LNCS. https://eprint.iacr.org/2015/565. Springer, 2015, pp. 214–235.

[Dül+15] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez and Peter Schwabe. "High-speed Curve25519 on 8-bit, 16-bit and 32-bit microcontrollers". In: *Design, Codes and Cryptography* 77.2 (2015). http://cryptojedi.org/papers/#mu25519.

[HS13] Michael Hutter and Peter Schwabe. "NaCl on 8-bit AVR Microcontrollers". In: *Progress in Cryptology – AFRICACRYPT 2013*. Ed. by Amr Youssef and Abderrahmane Nitaj. Vol. 7918. LNCS. `http://cryptojedi.org/papers/#avrnacl`. Springer, 2013, pp. 156–172.

[HS15] Michael Hutter and Peter Schwabe. "Multiprecision multiplication on AVR revisited". In: *Journal of Cryptographic Engineering* 5.3 (2015). `http://cryptojedi.org/papers/#avrmul`, pp. 201–214.

[LWG14] Zhe Liu, Erich Wenger and Johann Großschädl. "MoTE-ECC: Energy-Scalable Elliptic Curve Cryptography for Wireless Sensor Networks". In: *Applied Cryptography and Network Security*. Ed. by Ioana Boureanui, Philippe Owesarski and Serge Vaudenay. Vol. 8479. LNCS. `https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=77985`. Springer, 2014, pp. 361–379.

[NLD15] Erick Nascimento, Julio López and Ricardo Dahab. "Efficient and Secure Elliptic Curve Cryptography for 8-bit AVR Microcontrollers". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Rajat Subhra Chakraborty, Peter Schwabe and Jon Solworth. Vol. 9354. LNCS. Springer, 2015, pp. 289–309.

[WUW13] Erich Wenger, Thomas Unterluggauer and Mario Werner. "8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors". In: *Progress in Cryptology – INDOCRYPT 2013*. Ed. by Goutam Paul and Serge Vaudenay. Vol. 8250. LNCS. `https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=72486`. Springer, 2013, pp. 244–261.