# High-Performance Scalar Multiplication using 8-Dimensional GLV/GLS Decomposition

Joppe W. Bos, Craig Costello, Huseyin Hisil, Kristin Lauter

CHES 2013

Microsoft Research

YAŞAR UNIVERSITY

# Motivation - I

| | DH | ECDH |
|---|---|---|
| Group | $(\mathbf{F}_{p_1}^{*}, \times)$ | $(E(\mathbf{F}_{p_2}), +)$ |

| Security level (bits) | $\log_2 p_1$ | $\log_2 p_2$ | Ratio DH cost : EC cost |
|---|---|---|---|
| 128 | 3072 | 256 | 10:1 |
| 192 | 7680 | 384 | 32:1 |
| 256 | 15360 | 521 | 64:1 |

Source: NSA – The case for Elliptic Curve Cryptography
http://www.nsa.gov/business/programs/elliptic_curve.shtml

# Motivation - I

| | DH | ECDH |
|---|---|---|
| Group | $(\mathbf{F}_{p_1}^*, \times)$ | $(E(\mathbf{F}_{p_2}), +)$ |

| Security level (bits) | $\log_2 p_1$ | $\log_2 p_2$ | Ratio DH cost : EC cost |
|---|---|---|---|
| 128 | 3072 | 256 | 10:1 |
| 192 | 7680 | 384 | 32:1 |
| 256 | 15360 | 521 | 64:1 |

Source: NSA – The case for Elliptic Curve Cryptography
http://www.nsa.gov/business/programs/elliptic_curve.shtml

Can we do better?

# Motivation - II

## Can we do better?

Core ECC operation: **Elliptic curve scalar multiplication**

sequence of point additions and point doublings

# Motivation - II

## Can we do better?

Core ECC operation: **Elliptic curve scalar multiplication**
sequence of point additions and point doublings

### Reduce the **cost** of the group operation

- Use a different curve representation
- Use a different coordinate system
- E.g. Twisted Edwards curves with extended twisted Edwards coordinates
- See the Explicit-Formulas Database

# Motivation - II

## Can we do better?

Core ECC operation: **Elliptic curve scalar multiplication**
sequence of point additions and point doublings

### Reduce the **cost** of the group operation

- Use a different curve representation
- Use a different coordinate system
- E.g. Twisted Edwards curves with extended twisted Edwards coordinates
- See the Explicit-Formulas Database

### Reduce the **number** of group operations

- Different algorithms to compute the scalar multiplication
- Reduce the number of point additions:
  e.g. use large window sizes
- Reduce the number of point doublings:
  e.g. scalar decomposition

# Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* “mini-scalars” $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

# Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* “mini-scalars” $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\left([\lambda^{d-1}]P\right)$$

# Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\big([\lambda^{d-1}]P\big)$$

| | | | | |
|---|---|---|---|---|
| $k_0 =$ | $k_{0,0}$ | $k_{0,1}$ | $k_{0,2}$ | $k_{0,3}$ |
| $k_1 =$ | $k_{1,0}$ | $k_{1,1}$ | $k_{1,2}$ | $k_{1,3}$ |

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

# Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i\lambda^i]\,P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\big([\lambda^{d-1}]P\big)$$

$$k_0 = \quad \boxed{k_{0,0}}\ \boxed{k_{0,1}}\ \boxed{k_{0,2}}\ \boxed{k_{0,3}}$$

$$k_1 = \quad \boxed{k_{1,0}}\ \boxed{k_{1,1}}\ \boxed{k_{1,2}}\ \boxed{k_{1,3}}$$

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

# Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i\lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\left([\lambda^{d-1}]P\right)$$

$$\begin{array}{ll} k_0 = & \boxed{k_{0,0}}\ \boxed{k_{0,1}}\ \boxed{k_{0,2}}\ \boxed{k_{0,3}} \\ k_1 = & \boxed{k_{1,0}}\ \boxed{k_{1,1}}\ \boxed{k_{1,2}}\ \boxed{k_{1,3}} \end{array}$$

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

# Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\big([\lambda^{d-1}]P\big)$$

| | | | | |
|---|---|---|---|---|
| $k_0 =$ | $k_{0,0}$ | $k_{0,1}$ | $k_{0,2}$ | $k_{0,3}$ |
| $k_1 =$ | $k_{1,0}$ | $k_{1,1}$ | $k_{1,2}$ | $k_{1,3}$ |

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

# Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

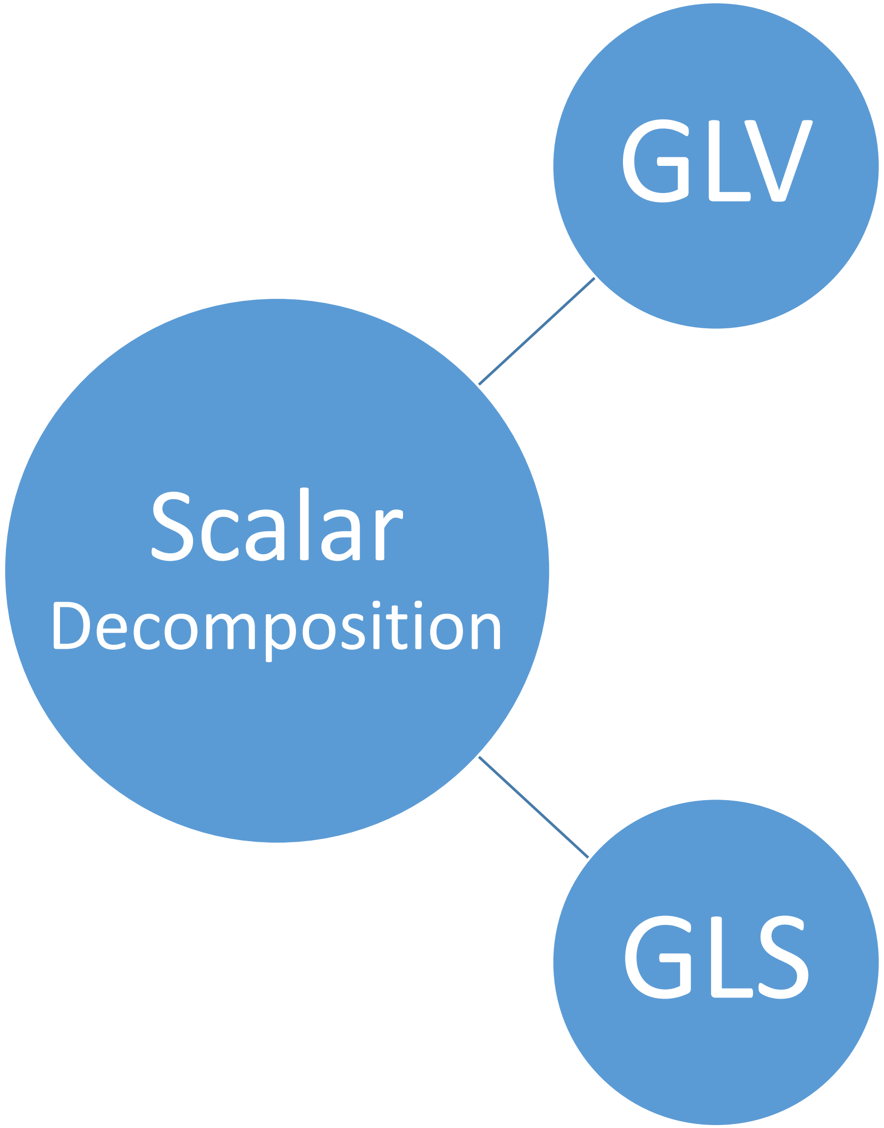
Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\left([\lambda^{d-1}]P\right)$$

| | | | | |
|---|---|---|---|---|
| $k_0 =$ | $k_{0,0}$ | $k_{0,1}$ | $k_{0,2}$ | $k_{0,3}$ |
| $k_1 =$ | $k_{1,0}$ | $k_{1,1}$ | $k_{1,2}$ | $k_{1,3}$ |

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

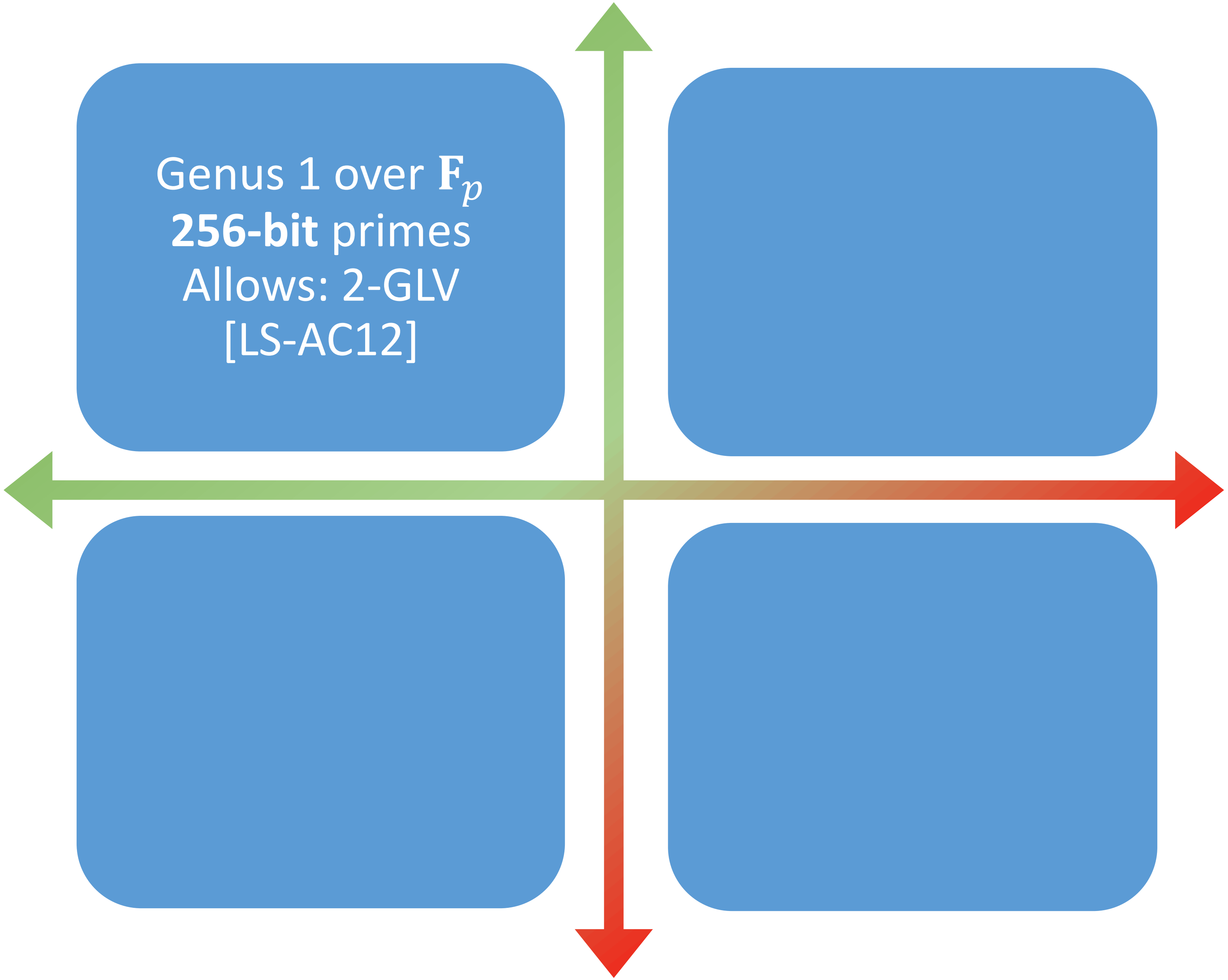
Example: $d = 2$



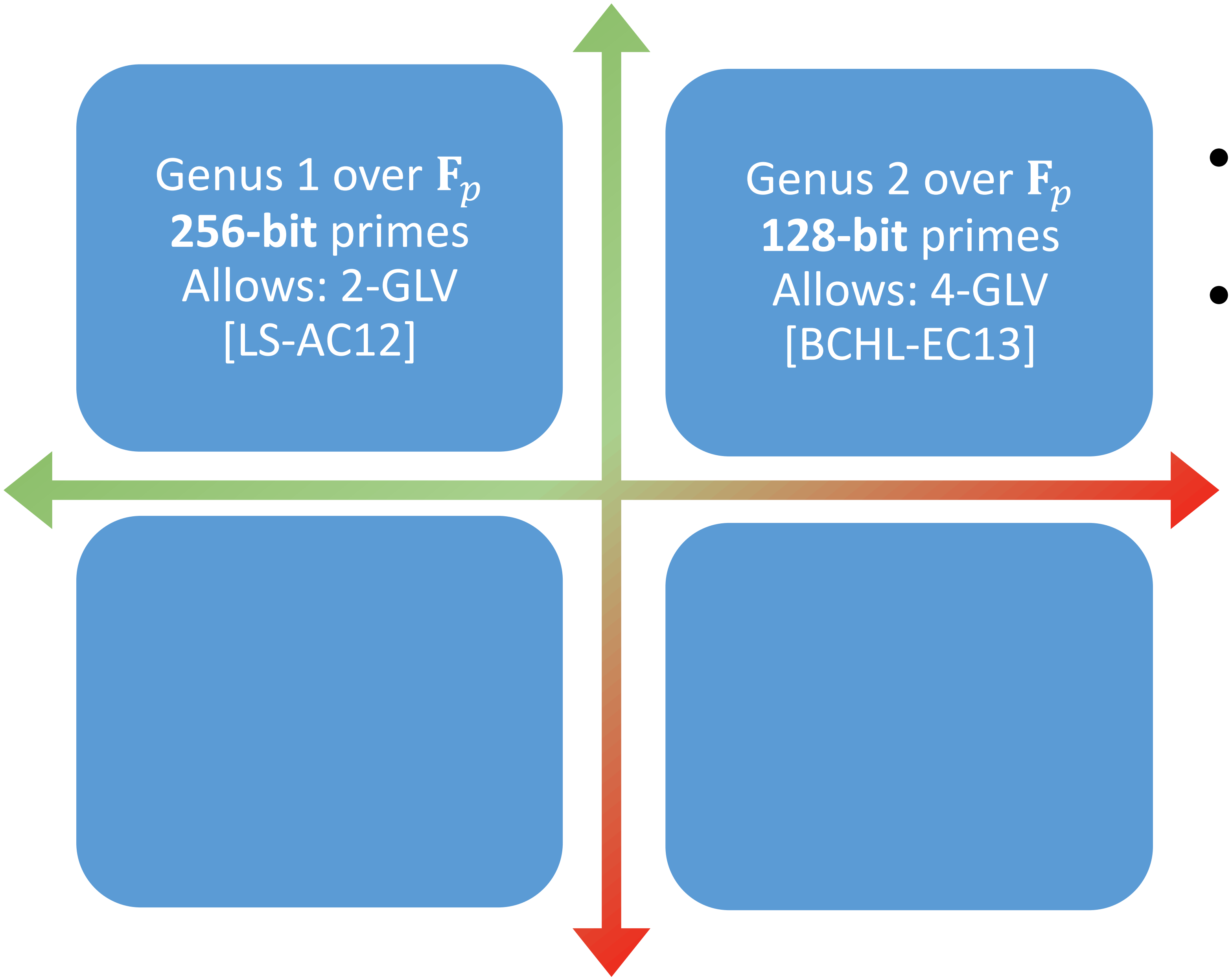

- Gallant, Lambert, Vanstone [GLV-C01]
- Use non-trivial endomorphism
- Larger endomorphism ring means larger dimensional scalar decomposition

- Galbraith, Lin, Scott [GLS-JoC11]
- Independent of the endomorphism
- Works for E($\mathbf{F}_{p^m}$) with $m > 1$

# Overview of GLV/GLS techniques – 128-bit security



# Overview of GLV/GLS techniques – 128-bit security

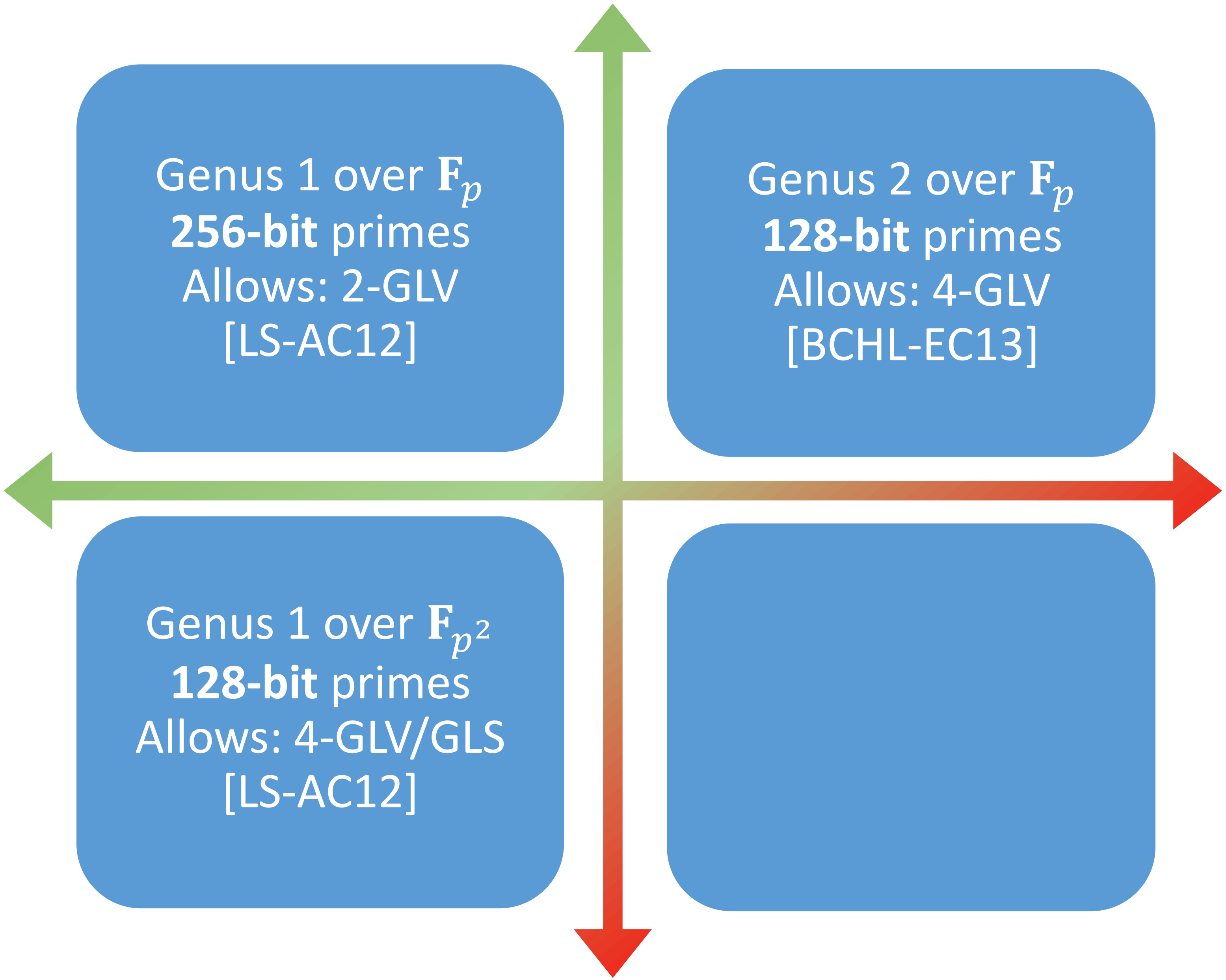



- Larger genus means larger endomorphism
- But genus > 3 is considered insecure

$\#E(\mathbf{F}_{p_1}) \approx \#\mathrm{Jac}_C(\mathbf{F}_{p_2})$ with

$$\log_2(p_1) \approx {}^1/_2 \log_2(p_2)$$

# Overview of GLV/GLS techniques – 128-bit security




$\#E\left(\mathbf{F}_{p_1}\right) \approx \#E\left(\mathbf{F}_{p_2^2}\right)$ with

$\log_2(p_1) \approx {}^{1}/_{2} \log_2(p_2^2)$

- Extension degree of *m*>1 means we can use GLS
- When *m* is too large subexponential attacks

# Overview of GLV/GLS techniques – 128-bit security

Genus 1 over $\mathbf{F}_p$
**256-bit** primes
Allows: 2-GLV
[LS-AC12]

Genus 2 over $\mathbf{F}_p$
**128-bit** primes
Allows: 4-GLV
[BCHL-EC13]

Genus 1 over $\mathbf{F}_{p^2}$
**128-bit** primes
Allows: 4-GLV/GLS
[LS-AC12]

?

- Larger genus means larger endomorphism
- But genus > 3 is considered insecure

$$\#E\left(\mathbf{F}_{p_1}\right) \approx \#\mathrm{Jac}_C\left(\mathbf{F}_{p_2^2}\right) \text{ with } \log_2(p_1) \approx ?\, \log_2(p_2^2)$$

- Extension degree of *m*>1 means we can use GLS
- When *m* is too large subexponential attacks or too slow

# Overview of GLV/GLS techniques – 128-bit security

Genus 1 over $\mathbf{F}_p$
**256-bit** primes
Allows: 2-GLV
[LS-AC12]

Genus 2 over $\mathbf{F}_p$
**128-bit** primes
Allows: 4-GLV
[BCHL-EC13]

Genus 1 over $\mathbf{F}_{p^2}$
**128-bit** primes
Allows: 4-GLV/GLS
[LS-AC12]

Genus 2 over $\mathbf{F}_{p^2}$
**64-bit** primes
Allows: 8-GLV/GLS

$\#E\left(\mathbf{F}_{p_1}\right) \approx \#\mathrm{Jac}_C\left(\mathbf{F}_{p_2^2}\right)$ with

$\log_2(p_1) \approx {}^1/_4 \log_2(p_2^2)$

Fits in a single word on high-end servers

Ideal candidate for embedded platforms

# Overview of GLV/GLS techniques – 128-bit security

Genus 1 over $\mathbf{F}_p$
**256-bit** primes
Allows: 2-GLV
[LS-AC12]

Genus 2 over $\mathbf{F}_p$
**128-bit** primes
Allows: 4-GLV
[BCHL-EC13]

Genus 1 over $\mathbf{F}_{p^2}$
**128-bit** primes
Allows: 4-GLV/GLS
[LS-AC12]

Genus 2 over $\mathbf{F}_{p^2}$
**64-bit** primes
Allows: 8-GLV/GLS

What about security?

How to efficiently use 8-GLV/GLS?

$\#E\left(\mathbf{F}_{p_1}\right) \approx \#\mathrm{Jac}_C\left(\mathbf{F}_{p_2^2}\right)$ with

$$\log_2(p_1) \approx {}^1/_4 \log_2(p_2^2)$$

Fits in a single word on high-end servers

Ideal candidate for embedded platforms

# Fast Primes – NIST-like and Montgomery Friendly

[BCHL-EC13] uses Mersenne prime: $2^{127} - 1$, we try: $2^{61} - 1$
Also the "NIST-like" prime: $2^{64} - 2285$

# Fast Primes – NIST-like and Montgomery Friendly

[BCHL-EC13] uses Mersenne prime: $2^{127} - 1$, we try: $2^{61} - 1$
Also the "NIST-like" prime: $2^{64} - 2285$

Interleaved radix-$2^b$ Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

for $i = 0$ to $n - 1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C + q \cdot p}{2^b}$$

# Fast Primes – NIST-like and Montgomery Friendly

[BCHL-EC13] uses Mersenne prime: $2^{127} - 1$, we try: $2^{61} - 1$
Also the "NIST-like" prime: $2^{64} - 2285$

Interleaved radix-$2^b$ Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

for $i = 0$ to $n - 1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C + q \cdot p}{2^b}$$

Not much we can do: this is the multiplication

# Fast Primes – NIST-like and Montgomery Friendly

[BCHL-EC13] uses Mersenne prime: $2^{127}-1$, we try: $2^{61}-1$
Also the "NIST-like" prime: $2^{64}-2285$

Interleaved radix-$2^b$ Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

for $i = 0$ to $n-1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C + q \cdot p}{2^b}$$

Not much we can do: this is the multiplication

If $p = \pm 1 \bmod 2^{\mathrm{b}}$ then $\mu = \mp 1 \bmod 2^{\mathrm{b}}$

# Fast Primes – NIST-like and Montgomery Friendly

[BCHL-EC13] uses Mersenne prime: $2^{127} - 1$, we try: $2^{61} - 1$
Also the "NIST-like" prime: $2^{64} - 2285$

Interleaved radix-$2^b$ Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

for $i = 0$ to $n - 1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C + q \cdot p}{2^b}$$

Not much we can do: this is the multiplication

If $p = \pm 1 \bmod 2^{\mathrm{b}}$ then $\mu = \mp 1 \bmod 2^{\mathrm{b}}$

Additionally, if $p$ has a "special" form: avoid muls

# Fast Primes – NIST-like and Montgomery Friendly

[BCHL-EC13] uses Mersenne prime: $2^{127}-1$, we try: $2^{61}-1$
Also the "NIST-like" prime: $2^{64}-2285$

$= 2^{32}(2^{29}-0)-1$

Interleaved radix-$2^b$ Montgomery multiplication

Example: $2^b\left(2^{\tilde{b}}-c\right)-1$

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

for $i = 0$ to $n-1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C + q \cdot p}{2^b}$$

Not much we can do: this is the multiplication

If $p = \pm 1 \bmod 2^{\mathrm{b}}$ then $\mu = \mp 1 \bmod 2^{\mathrm{b}}$

Additionally, if $p$ has a "special" form: avoid muls

# Security of Genus 2 curves over $\mathbf{F}_{p^2}$ with Group Order $h \cdot r$

## Generic Attack: Pollard rho

- [Pollard-MoC78]
- $\sqrt{(\pi r)/(2\#\mathrm{Aut})}$, where $\#\mathrm{Aut} \geq 2$
- For the GLV/GLS curves $\#\mathrm{Aut} = 10$

## Weil Descent and Index Calculus

- [P. Gaudry. J. Symb. Comp. 09], [K. Nagao. ANTS 10]

$$\tilde{O}\left(p^{2-2/ng}\right)$$

- For a **fixed** extension degree *n* and genus *g*

# Security of Genus 2 curves over $\mathbf{F}_{p^2}$ with Group Order $h \cdot r$

## Generic Attack: Pollard rho

- [Pollard-MoC78]
- $\sqrt{(\pi r)/(2\#\mathrm{Aut})}$, where $\#\mathrm{Aut} \geq 2$
- For the GLV/GLS curves $\#\mathrm{Aut} = 10$

## Weil Descent and Index Calculus

- [P. Gaudry. J. Symb. Comp. 09], [K. Nagao. ANTS 10]

$$O\left(p^{2-2/ng} \log(p)^{s}\right)$$

- For a **fixed** extension degree *n* and genus *g*

# Security of Genus 2 curves over $\mathbf{F}_{p^2}$ with Group Order $h \cdot r$

## Generic Attack: Pollard rho

- [Pollard-MoC78]
- $\sqrt{(\pi r)/(2\#\mathrm{Aut})}$, where $\#\mathrm{Aut} \geq 2$
- For the GLV/GLS curves $\#\mathrm{Aut} = 10$

## Weil Descent and Index Calculus

- [P. Gaudry. J. Symb. Comp. 09], [K. Nagao. ANTS 10]

$$O\left(p^{2-2/ng} \log(p)^s \, 2^{3n(n-1)g}\right)$$

- For a **fixed** extension degree *n* and genus *g*

# Security of Genus 2 curves over $\mathbf{F}_{p^2}$ with Group Order $h \cdot r$

## Generic Attack: Pollard rho

- [Pollard-MoC78]
- $\sqrt{(\pi r)/(2\#\mathrm{Aut})}$, where $\#\mathrm{Aut} \geq 2$
- For the GLV/GLS curves $\#\mathrm{Aut} = 10$

## Weil Descent and Index Calculus

- [P. Gaudry. J. Symb. Comp. 09], [K. Nagao. ANTS 10]

$$O(p^{2-2/ng} \log(p)^s \, 2^{3n(n-1)g})$$

- For a **fixed** extension degree *n* and genus *g*
- Take $g = n = 2, s = 1$

$$p^{3/2} \cdot \log(p) \cdot 2^{12}$$

# Security of Genus 2 curves over $\mathbf{F}_{p^2}$ with Group Order $h \cdot r$

## Generic Attack: Pollard rho

- [Pollard-MoC78]
- $\sqrt{(\pi r)/(2\#\mathrm{Aut})}$, where $\#\mathrm{Aut} \geq 2$
- For the GLV/GLS curves $\#\mathrm{Aut} = 10$

## Weil Descent and Index Calculus

- [P. Gaudry. J. Symb. Comp. 09], [K. Nagao. ANTS 10]

$$O\left(p^{2-2/ng} \log(p)^s \, 2^{3n(n-1)g}\right)$$

- For a **fixed** extension degree *n* and genus *g*
- Take $g = n = 2, s = 1$

$$p^{3/2} \cdot \log(p) \cdot 2^{12}$$

| $p$ | $\log_2 h$ | $\log_2 r$ | Security rho | Security i.c. |
|---|---|---|---|---|
| $2^{61} - 1$ | 32 | 213 | 105 | 109 |
| $(2^{31}-201) \cdot 2^{32} - 1$ | 31 | 222 | 109 | 112 |
| $2^{64} - 2285$ | 33 | 224 | 111 | 113 |

# Security of Genus 2 curves over $\mathbf{F}_{p^2}$ with Group Order $h \cdot r$

## Generic Attack: Pollard rho

- [Pollard-MoC78]
- $\sqrt{(\pi r)/(2\#\mathrm{Aut})}$, where $\#\mathrm{Aut} \geq 2$
- For the GLV/GLS curves $\#\mathrm{Aut} = 10$

## Weil Descent and Index Calculus

- [P. Gaudry. J. Symb. Comp. 09], [K. Nagao. ANTS 10]

$$O(p^{2-2/ng} \log(p)^s \, 2^{3n(n-1)g})$$

- For a **fixed** extension degree *n* and genus *g*
- Take $g = n = 2, s = 1$

$$p^{3/2} \cdot \log(p) \cdot 2^{12}$$

**Advantage**
Smaller scalars

| $p$ | $\log_2 h$ | $\log_2 r$ | Security rho | Security i.c. |
|---|---|---|---|---|
| $2^{61} - 1$ | 32 | 213 | 105 | 109 |
| $(2^{31}-201) \cdot 2^{32} - 1$ | 31 | 222 | 109 | 112 |
| $2^{64} - 2285$ | 33 | 224 | 111 | 113 |

# 8-Dimensional GLV/GLS

- Buhler-Koblitz curves: $C/\mathbf{F}_{p^2} : \; y^2 = x^5 + a$
- $\psi$: $\mathrm{Jac}(C) \rightarrow \mathrm{Jac}(C), \; \psi(D) = [\lambda]D$, for $0 < \lambda < r$
- Decompose the scalar using [Park,Jeong,Lim-EC02]

Lookup table: $L[i] = \sum_{\ell=0}^{7}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_\ell$ for $0 \leq i < 2^8$

# 8-Dimensional GLV/GLS

- Buhler-Koblitz curves: $C/\mathbf{F}_{p^2} : \; y^2 = x^5 + a$
- $\psi$: $\mathrm{Jac}(C) \to \mathrm{Jac}(C)$, $\psi(D) = [\lambda]D$, for $0 < \lambda < r$
- Decompose the scalar using [Park,Jeong,Lim-EC02]

Naive
$32\mathrm{D} + (256 + 32)\mathrm{A}$

Lookup table: $$L[i] = \sum_{\ell=0}^{7}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_\ell \text{ for } 0 \le i < 2^8$$

# 8-Dimensional GLV/GLS

- Buhler-Koblitz curves: $C/\mathbf{F}_{p^2}:\ y^2 = x^5 + a$
- $\psi$: Jac$(C) \rightarrow$ Jac$(C)$, $\psi(D) = [\lambda]D$, for $0 < \lambda < r$
- Decompose the scalar using [Park,Jeong,Lim-EC02]

Naive
$32\mathrm{D} + (256 + 32)\mathrm{A}$

Lookup table: $L[i] = \sum_{\ell=0}^{7}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_\ell$ for $0 \leq i < 2^8$

Two lookup tables:

$$T_1[i] = \sum_{\ell=0}^{3}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_\ell$$

$$T_2[i] = \sum_{\ell=0}^{3}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_{\ell+4}$$

for $0 \leq i < 2^4$

# 8-Dimensional GLV/GLS

- Buhler-Koblitz curves: $C/\mathbf{F}_{p^2}:\ y^2 = x^5 + a$
- $\psi$: $\mathrm{Jac}(C) \to \mathrm{Jac}(C)$, $\psi(D) = [\lambda]D$, for $0 < \lambda < r$
- Decompose the scalar using [Park,Jeong,Lim-EC02]

Naive
$32\mathrm{D} + (256 + 32)\mathrm{A}$

Lookup table: $L[i] = \sum_{\ell=0}^{7}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_\ell$ for $0 \leq i < 2^8$

Two table estimate
$32\mathrm{D} +$
$(2 \times 16 + 2 \times 32)\mathrm{A}$

Two lookup tables:

$$T_1[i] = \sum_{\ell=0}^{3}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_\ell$$

$$T_2[i] = \sum_{\ell=0}^{3}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_{\ell+4}$$

for $0 \leq i < 2^4$

# 8-Dimensional GLV/GLS

- Buhler-Koblitz curves: $C/\mathbf{F}_{p^2}:\ y^2 = x^5 + a$
- $\psi$: $\mathrm{Jac}(C) \to \mathrm{Jac}(C)$, $\psi(D) = [\lambda]D$, for $0 < \lambda < r$
- Decompose the scalar using [Park,Jeong,Lim-EC02]

Naive
$32\mathrm{D} + (256 + 32)\mathrm{A}$

Lookup table: $L[i] = \sum_{\ell=0}^{7}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_\ell$ for $0 \le i < 2^8$

**Constant** time
$32\mathrm{D} + 7\psi$
$(22 + 2 \times 32)\mathrm{A}$

Two table estimate
$32\mathrm{D} +$
$(2 \times 16 + 2 \times 32)\mathrm{A}$

Two lookup tables:

$$T_1[i] = \sum_{\ell=0}^{3}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_\ell$$

$$T_2[i] = \sum_{\ell=0}^{3}\left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2\right) \cdot D_{\ell+4}$$

for $0 \le i < 2^4$

**Non-constant** time
$32\mathrm{D} + 16\psi$
$(13 + 2 \times 32)\mathrm{A}$

# Performance Results – x86

**Platform**: Intel Core i7-3520M Ivy Bridge (2893.484 MHz), hyperthreading turned off and over-clocking (“turbo boost”) disabled

We didn’t aim for record-performance.
There is room for improvement!

| Reference | $(g, \mathrm{K})$ | CT | Bit sec | $10^3$ cycles |
|---|---|---|---|---|
| NIST-p224 | $(1, \mathbf{F}_p)$ | | 112 | 302 |
| [B-PKC06] curve25519 | $(1, \mathbf{F}_p)$ | | 126 | 182 |
| [FPH-ep13] 4-GLV/GLS | $(1, \mathbf{F}_{p^2})$ | | 125 | 92 |
| [BCHL-EC13] Kummer | $(2, \mathbf{F}_p)$ | | 125 | 117 |
| [BCHL-EC13] 4-GLV | $(2, \mathbf{F}_p)$ | | 125 | 156 |
| $2^{61} - 1$, Kummer | $(2, \mathbf{F}_{p^2})$ | | 103 | 108 |
| $2^{61} - 1$, 8-GLV/GLS | $(2, \mathbf{F}_{p^2})$ | | 105 | 100 (88) |

- 3x faster than NIST
- Slightly less secure

# Performance Results – x86

**Platform**: Intel Core i7-3520M Ivy Bridge (2893.484 MHz), hyperthreading turned off and over-clocking ("turbo boost") disabled

We didn't aim for record-performance.
There is room for improvement!

| Reference | $(g, \mathrm{K})$ | CT | Bit sec | $10^3$ cycles |
|---|---|---|---|---|
| NIST-p224 | $(1, \mathbf{F}_p)$ | | 112 | 302 |
| [B-PKC06] curve25519 | $(1, \mathbf{F}_p)$ | | 126 | 182 |
| [FPH-ep13] 4-GLV/GLS | $(1, \mathbf{F}_{p^2})$ | | 125 | 92 |
| [BCHL-EC13] Kummer | $(2, \mathbf{F}_p)$ | | 125 | 117 |
| [BCHL-EC13] 4-GLV | $(2, \mathbf{F}_p)$ | | 125 | 156 |
| $2^{61} - 1$, Kummer | $(2, \mathbf{F}_{p^2})$ | | 103 | 108 |
| $2^{61} - 1$, 8-GLV/GLS | $(2, \mathbf{F}_{p^2})$ | | 105 | 100 (88) |

- Similar performance
- 105 vs 126 bits security

# Performance Results – ARM

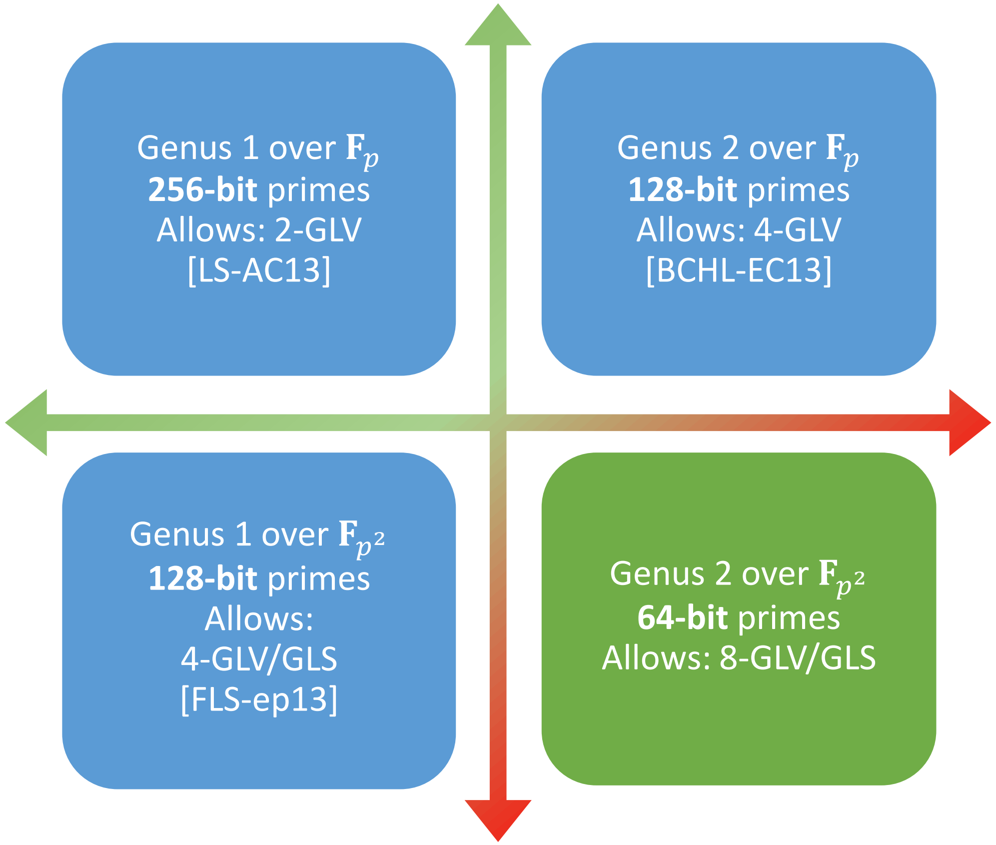
**Platform**: BeagleBoard-xM (1 GHz Cortex-A8 ARM core)
Use the Montgomery-friendly arithmetic

We didn't aim for record-performance.
There is room for improvement!

- Order of magnitude faster than NIST-p224
- Our performance in the same ballpark
- Lower security level

| | Platform | $(g, \mathrm{K})$ | CT | Bit sec | $10^3$ cycles |
|---|---|---|---|---|---|
| [MTS-SASP11] NIST-p224 | Cortex-A8 | $(1, \mathbf{F}_p)$ | | 112 | 7805 |
| [BS-CHES12] curve25519 | Cortex-A8 w NEON | $(1, \mathbf{F}_p)$ | | 126 | 527 |
| [FPH-ep13] 4-GLV/GLS | Cortex-A9 | $(1, \mathbf{F}_{p^2})$ | | 125 | 417 |
| [H-ep12] twisted Edwards | Cortex-A9 | $(1, \mathbf{F}_p)$ | | 125 | 616 |
| $2^{61}-1$, Kummer | Cortex-A8 | $(2, \mathbf{F}_{p^2})$ | | 103 | 767 |
| $2^{61}-1$, 8-GLV/GLS | Cortex-A8 | $(2, \mathbf{F}_{p^2})$ | | 105 | 617 (576) |

# Conclusions




- ✓ Genus 2 over $\mathbf{F}_{p^2}$ allows to work with 64-bit primes
- ✓ Interesting for both high-end 64-bit servers and embedded 32-bit devices
- ✓ Precomputing the lookup table for 8-GLV/GLS is more involved than for 2- and 4-GLV/GLS
- ✓ Although faster attacks exist, still provides sufficient security

See our full paper:
Cryptology ePrint Archive: **Report 2013/146**
The x86 implementations have been submitted to eBACS