Modern Elliptic Curve Cryptography for Constrained Devices

Project Report — PR2

Tobias Markmann
tobias.markmann@haw-hamburg.de

April 9, 2015

Constrained devices see wide application in wireless sensor network (WSN) and more recently the Internet of Things (IoT). Security is an important aspect of these networks as nodes communicate over open wireless channels or via the insecure global Internet. Elliptic curve cryptography (ECC) provides a cryptographic basis for modern security protocols by requiring smaller keys and faster operation compared to classic RSA cryptosystems.

This report covers the implementation of twisted Edwards curves — a simpler and faster form of curves for ECC — to the RELIC library. Two coordinate formats are supported by our implementation, standard projective and extended projective coordinates. We test the performance of our implementation in three benchmarks, a low-level ECC microbenchmark, an Elliptic Curve Diffie-Hellman (ECDH) macrobenchmark and a high-level benchmark of an identity-based signature (IBS) signature. These tests are carried out on a high-end desktop, an embedded and a constrained hardware platform.

We show that our implementation of the twisted Edwards curve can provide improvements of up to 31% on our embedded platforms for the ECC point multiplication, when compared to the existing short Weierstrass implementation in RELIC.

Contents

1 Introduction

With Internet of Things (IoT) adoption on the rise many questions open up regarding how to secure these networks and devices. Elliptic curve cryptography (ECC) is the foundation of many current asymmetric security mechanisms.

In the beginning of 2014, the Crypto Forum of the Internet Research Task Force (IRTF) began to work on defining a set of standard elliptic curves for various security levels to provide security to the users of the world wide web (WWW).

ECC provides better scalability with an increasing security level and at normal security levels for global communication, it outperforms classic asymmetric algorithms like RSA [1] cryptosystems and the Digital Signature Algorithm (DSA). To increase the adoption of ECC in the IoT and its constrained devices we aim to improve the performance of the ECC algorithms in the open source RELIC [2] C library.

In our previous work [3] we looked at various identity-based signature (IBS) schemes for application in IoT scenarios. The analysis covered an overview of IBSs and how they compare to classic public-key cryptography (PKC) with respect to securing communication patterns in the IoT. Performance measurements of three different IBSs showed ECC as a promising building block for asymmetric signature schemes due to their good balance between scalability in security and performance on different hardware architectures.

This report continues the work by showing the performance benefits of modern ECC with an implementation of a twisted Edwards [4] curve in the RELIC [2] C library.

We test our implementation of twisted Edwards curves in standard projective and extended projective coordinates on a wide range of hardware architectures. Our test evaluates the performance of our implementation for its basic ECC group operations and when used in basic protocols like Elliptic Curve Diffie-Hellman (ECDH). Further we evaluate our new implementation when used in conjunction with vBNN-IBS [5], an ECC-based IBS. IBS allow to use already existing information like IPv6 addresses as public keys and are an attractive option for securing networks where low communication overhead is important.

This report is structured as follows. Section 2 looks at related work in this area, specifically existing benchmark results of the scientific community and optimizations of ECC algorithms for highly constrained devices. Section 3 briefly covers background of Edwards and twisted Edwards curves and their common point representations. In Section 4 we describe the addition formulas and algorithms used in our implementation followed by an evaluation using benchmarks on three different hardware platforms in Section 5. We close with conclusions and a short outlook in future research opportunities in Section 6.

1.1 Operations and Nomenclature

The computational complexity of formulas for operations on elliptic curves (e.g. addition, doubling, etc.) is commonly specified in terms of operations in the underlying field. The following notation is used within this paper.

Let $n = \log_2 q$, i.e. the number of bits required to store q, for the finite field $\mathbb{F}_q$.

Notation	Description
I	Inversion
M	Multiplication
S	Squaring
D	Mulitplication with constant
add	Addition

Table 1: Notation used for describing the complexity of ECC operations.

Additions and subtractions (add) in $\mathbb{F}_q$ have the lowest cost. They can be performed in linear time complexity $\mathcal{O}(n)$.

General multiplication (M) has quadratic time complexity $\mathcal{O}(n^2)$ and is commonly implemented using Montgomery's multiplication.

Special cases like constant multiplication (D) and squaring (S) are faster than the general case. D has a lower complexity than S. However, the ratio of cost between M, S and D varies between different hardware platforms.

Inversions (I) are undoubtedly the most expensive operations due to their common implementation using the extended Euclidian algorithm. A naive implementation has a computational complexity of $\mathcal{O}(n^3)$. This can be further optimized but it will still be slower than general multiplication.

More detailed information can be found in [6, Chapter 2 and 14].

2 Related Work

In this section we focus on three different works related to implementing and benchmarking modern elliptic curve cryptography (ECC) on constrained devices.

eBACS [7] is a widely accepted resource on benchmark results for common cryptographic operations. The project assembles benchmarks for various primitives like *eBATS* for asymmetric cryptosystems, *eBASC* for stream ciphers and *eBASH* for cryptographic hash functions.

The *eBATS* benchmark are of particular interest to us, as they cover asymmetric signatures and protocols like Diffie-Hellman key exchange. While they include measurements for Curve25519 [8], they are using the Montgomery curve with an X-coordinate ladder [9] to compute Elliptic Curve Diffie-Hellman (ECDH) shared secrets. In contrast, the code described in this report and existing short Weierstrass curve code in RELIC uses full projective and extended projective coordinates for its computations.

Furthermore, *eBATS* does currently not cover identity-based signatures (IBSs) at all, which leaves ECDH as the sole algorithm for any comparison.

While the benchmarks of our implementation are not directly comparable to the various benchmarks of implementations in *eBATS*, it provides a good estimate on the absolute performance gap between our implementation and the top performing ECDH implementations on a particular architecture.

Optimization of modern ECC using Curve25519 and ED25519 for latest mobile ARM microcontrollers was evaluated by Bernstein *et al.* [10]. They achieve a performance of 527,102 cycles for an ECDH shared secret computation on a Cortex A8 chip. However, the Cortex A8 ARM microcontroller has a much higher clock frequency (1 GHz) compared to the Cortex M3 (186 MHz). Furthermore the Cortex A8 supports complex instructions like NEON vector instructions and the ECC implementation has optimized assembler to take advantage of the vector instructions. The highly constrained and low-energy Cortex M3 microcontroller is missing support for vector instructions.

Even though Cortex A8 and Cortex M3 are both using the ARM architecture there is a huge difference in the specific design and features of both microcontrollers. The Cortex A8 is primarily used in mobile phones and tables which come with a rather large battery compared to Internet of Things (IoT) devices and are regularly charged. Achieving a similar performance is out of reach due to the large differences in features and performance between the Cortex A8 and the Cortex M3 used in our work. However, it is important to note that Bernstein *et al.* also use a X-coordinate only Montgomery ladder for the shared secret ECDH computation.

de Clercq *et al.* [11] focused on improving the performance of ECC on ultra low-power platforms, in their case the ARM Cortex-M0+. They compared variable-base and fixed-based scalar multiplication for ECC of various implementations, including RELIC [2]. In addition, they proposed a new field multiplication algorithm for binary fields based on the *López-Dahab* algorithm [12]. By using *fixed registers* this new algorithm performs 15% better compared to a similar algorithm with *rotating registers*. When combined with scalar multiplication algorithms for elliptic curves, their variable-based scalar multiplication outperforms other software-based implementations with regard to energy use by a factor of at least 3.0.

However, their analysis and their newly proposed algorithms only work for binary fields and ECC using binary fields. Our work focuses on prime fields and performance improvements of ECC based on prime fields as they have a more stable security history. Nevertheless, their *"López-Dahab with fixed registers"* algorithm would be an interesting candidate to combine with Edwards curves defined over binary fields, also known as binary Edwards curves [13].

3 Background

In 2007 Edwards [4] proposed a new form of elliptic curves over number fields that are defined by the equation $x^2 + y^2 = c^2(1 + x^2y^2)$.

All elliptic curves over non-binary finite fields are transformable into the Edwards form of elliptic curves. However this transformation sometimes requires the Edwards form to be defined over a field extension of the original field [14].

Building on Edwards proposal, Bernstein *et al.* [14] defined an expanded formula for Edwards curves to include more curves for possible transformation to Edwards curves without change of the underlying finite field. Their formula for Edwards curves is $x^2+y^2 = c^2(1+dx^2y^2)$ where $cd\,(1-dc^4) \neq 0$ holds.

They propose a formula for Edwards curves to include all curves $x^2+y^2 = 1+dx^2y^2$ and proof that all elliptic curves with a point of order 4[1] are transformable to this Edwards curve formula. The addition law for Edwards curves is *unified*, meaning that it can be used for both addition and doubling. It is also *complete*, meaning it is valid for all possible inputs, including the identity element. Neither of these properties apply to classic Weierstrass curves and implementations need to handle special cases. Bernstein *et al.* also presented fast addition and doubling formulas and showed their advantage for the performance of elliptic curve cryptography (ECC).

Having a unified and complete addition law not only enables compact implementations but also reduces the attack surface on side-channels. ECC implementations using classic Weierstrass curves commonly have a highly branched addition law handling various special cases. Since not all branches are of equal computational complexity, the implementation is subject to simple power analysis (SPA), timing attacks and other side-channel attacks.

Following up, Bernstein *et al.* [16] generalized the Edwards curve formula even further and introduced twisted Edwards curves. They are a generalization of the Edwards curve formula and defined as $ax^2+y^2 = 1+dx^2y^2$ with $a, x, y, d \in \mathbb{F}_p$.

As a generalization, an increasing amount of elliptic curves in Weierstrass form can be transformed into twisted Edwards curves. Essentially all twisted Edwards curves can be written as Montgomery [9] curves and vice versa [16, section 3].

Furthermore, the Edwards curve $E_{E,1,(d/a)} : \bar{x}^2+\bar{y}^2 = 1+(d/a)\bar{x}^2\bar{y}^2$ is isomorphic to the twisted Edwards curve $E_{E,a,d} : ax^2+y^2 = 1+dx^2y^2$. If a is a square in the finite field of $E_{E,1,(d/a)}$ then its isomorphic twisted Edwards curve also exists in the same finite field. Else it only exists in an extension of the field. The points can be transfered from the Edwards curve group to the twisted Edwards curve group using the map $(x, y) = (\bar{x}/\sqrt{a}, \bar{y})$ [16, section 3].

This isomorphism already shows one possible performance advantage of twisted Edwards curves. For existing Edwards curves with the $\bar{d}$ parameter — also representable as (d/a) with a being a square — it can be a computational advantage to work with the isomorphic twisted Edwards curve instead.

An ECC group operation like point addition requires a multiplication with the $\bar{d}$ constant. The same operation for twisted Edwards curves uses two multiplications by d and a instead. In cases where the multiplications by d and a are cheaper than a multiplication by (d/a) it is preferable to perform computations on the twisted Edwards curve instead [16, section 7].

This shows that unified, complete and fast ECC addition formulas, i.e. the twisted Edwards addition formulas, are available for a wider range of curves. For details on the

[1]The order of point P is the smallest positive integer x with $x \cdot P = \mathcal{O}$, with $\mathcal{O}$ being the identity element of the elliptic curve group [15, p. 20].

number of Edwards and twisted Edwards curves over a finite field $\mathbb{F}_p$ see [16, section 4].

Later on Hisil *et al.* [17] suggested another point representation for twisted Edwards curves using a forth auxiliary coordinate, T, in addition to the three standard projective coordinates. While the curve formula remains the same, points are now represented as (X, Y, T, Z) with $T = \frac{XY}{Z}$. Further details on the point representation are described in [17, p. 330].

The advantage of these extended coordinates lies in the addition formula associated with them. The addition formula for the extended coordinates saves one multiplication and one squaring, providing a significant further performance improvement. However, this comes at the cost of a performance drop for the doubling formula. The doubling formula for extended twisted Edwards curves is one multiplication operation more expensive than the doubling formula for standard projective coordinates. See Table 2 and Table 3 for a direct cost comparison.

With a mixed coordinate scalar multiplication algorithm however there is still a performance advantage overall [17, p. 337] and the formulas for extended coordinates allow for heavy parallelization on multi-processor systems.

4 Implementing Edwards Curves for Constrained Devices

RELIC [2] is an open source cryptographic C library targeting at constrained devices. This specialization on constrained devices is reflected in the lightweight design and modularization. Furthermore, some low-level primitives come with optimized assembly instructions for architectures common in constrained environments like wireless sensor network (WSN) or the Internet of Things (IoT).

RELIC already provides elliptic curves over prime and over binary fields, supporting both affine and projective point representations.

RELIC implements the following algorithms for general scalar multiplications on elliptic curves:

1. Binary method [15, p. 146] (`BASIC`)
2. Sliding window method [15, p. 149] (`SLIDE`)
3. Montgomery's ladder [15, p. 287] (`MONTY`)
4. Left-to-right window NAF method [15, p. 153] (`LWNAF`)

Due to the current design of RELIC the existing algorithms for scalar multiplication in an elliptic curve group could not directly be reused. However, the scalar multiplication algorithms could easily be duplicated from the short Weierstrass curve prime field implementation and adapted to our new twisted Edwards curve implementation. In addition our twisted Edwards curve implementation takes advantage of the RELIC prime field implementation.

The next three sections will introduce details of the implementation of twisted Edwards curve, the extended coordinate format for twisted Edwards curves, and finally the mixed coordinate scalar multiplication.

4.1 Implementation of Twisted Edwards Curves

We implement twisted Edwards curves using projective coordinates to reduce the number of expensive inversions in $\mathbb{F}_p$ for operations on the elliptic curve. This is a wide-spread approach and can also be seen in the existing RELIC implementation for short Weierstrass curves and other elliptic curve cryptography (ECC) libraries.

Our implementation is part of a new RELIC module for twisted Edwards curves, the `ED` module. The curve formula for twisted Edwards curves using projective coordinates is $(aX^2 + Y^2)\,Z^2 = Z^4 + dX^2Y^2$ with points as (X_1, Y_1, Z_1) being equivalent to the affine point $\left(\frac{X_1}{Z_1}, \frac{Y_1}{Z_1}\right)$. We implement twisted Edwards curves for lightweight ECC as described by Bernstein *et al.* [16].

The formulas for point addition and doubling arithmetic in the twisted Edwards elliptic curve group used by our implementation are as follows.

The point addition $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, Z_2)$ is computed using the following formulas:

$$
\begin{aligned}
X_3 &= A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D) && \text{with } A = Z_1 \cdot Z_2 \\
Y_3 &= A \cdot G \cdot (D - aC) && F = B - E \quad \text{with } B = A^2 \\
Z_3 &= F \cdot G && C = X_1 + X_2 \quad E = dC \cdot D \\
& && D = Y_1 \cdot Y_2 \\
& && G = B + E
\end{aligned}
$$

These formulas sum up to a computational complexity of $10\text{M} + 1\text{S} + 2\text{D} + 7\text{add}$ [16, p. 12].

The point doubling $(X_3, Y_3, Z_3) = 2 \cdot (X_1, Y_1, Z_1)$ is computed using the following formulas:

$$
\begin{aligned}
X_3 &= (B - C - D) \cdot J && \text{with } B = (X_1 + Y_1)^2 \\
Y_3 &= F \cdot (E - D) && C = {X_1}^2 \\
Z_3 &= F \cdot J && D = {Y_1}^2 \\
& && E = aC \\
& && F = E + D \\
& && J = F - 2H \quad \text{with } H = {Z_1}^2
\end{aligned}
$$

These formulas sum up to a computational complexity of $3\text{M} + 4\text{S} + 1\text{D} + 7\text{add}$ [16, p. 12].

In the following, we compare the costs of these formulas with the addition formulas already implemented in RELIC. We compare the computational costs of the existing short

Weierstrass curve implementation in RELIC with our new twisted Edwards curve implementation.

Table 2 shows the computational costs for addition of two elliptic curve points for the existing short Weierstrass curve implementations in the top half of the table and our new twisted Edwards curve implementations in the lower half. The computational costs are described as basic operations required in the underlying finite field for the point addition. The basic operations are explained in Table 1.

Curve	Coordinates	Cost in $\mathbb{F}_p$
Short Weierstrass	Affine	$1\text{I} + 2\text{M} + 1\text{S} + 6\text{add}$
Short Weierstrass	Projective	$11\text{M} + 5\text{S} + 9\text{add}$
Twisted Edwards	Projective	$10\text{M} + 1\text{S} + 2\text{D} + 7\text{add}$
Twisted Edwards	Extended Projective	$9\text{M} + 2\text{D} + 7\text{add}$

Table 2: Computational complexity of point addition methods implemented in RELIC for Weierstrass and Edwards curves. Existing implementations at the top; new Edwards curve implementations at the bottom.

Table 3 compares the cost for point doubling. The interesting observation here is between the projective and extended projective coordinates for twisted Edwards curves. At a first glance the extended projective coordinates are 1M operation more expensive. However, this multiplication is used for the T coordinate which is not required for the doubling method and thus can be skipped during multiple consecutive doublings and calculated once at the end.

Curve	Coordinates	Cost in $\mathbb{F}_p$
Short Weierstrass	Affine	$1\text{I} + 2\text{M} + 2\text{S} + 8\text{add}$
Short Weierstrass	Projective	$3\text{M} + 5\text{S} + 8\text{add}$
Twisted Edwards	Projective	$3\text{M} + 4\text{S} + 1\text{D} + 7\text{add}$
Twisted Edwards	Extended Projective	$4\text{M} + 4\text{S} + 1\text{D} + 7\text{add}$

Table 3: Computational complexity of point doubling methods implemented in RELIC for Weierstrass and Edwards curves. Existing implementations at the top; new Edwards curve implementations at bottom.

It is worth noting that the cost for the twisted Edwards curve operations describe *complete*, e.g. constant time, implementation that does not need to handle special cases. This reduces possible side-channels. To make short Weierstrass operations similarly safe against side-channel attacks would require additional code which increases code complexity and may introduce performance penalties.

As a first example, we implement the twisted Edwards curve ED22519, defined as $E(\mathbb{F}_p)$: $ax^2 + y^2 = 1 + dx^2y^2$ with $a = -1, d = -\frac{121665}{121666}$ and $p = 2^{255} - 19$. This curve was introduced by Bernstein *et al.* [18] in their crypto system for high-speed and high-security asymmetric signatures. The equivalent short Weierstrass curve is already implemented in RELIC.

Further twisted Edwards curves can easily be added to RELIC by specifying its parameters, i.e. the curve parameters a and d, the prime of the field, the base or generator point with its x- and y-coordinate and the cofactor of the elliptic curve group h.

4.2 Implementation of Extended Coordinates

We followed up with an implementation of extended twisted Edwards coordinates [17] and their addition and doubling formulas. These coordinates further reduce the cost of ECC point additions in exchange for more complex scalar multiplication methods.

The formulas used for point addition and doubling for extended coordinates are the following.
The point addition $(X_3, Y_3, T_3, Z_3) = (X_1, Y_1, T_1, Z_1) + (X_2, Y_2, T_2, Z_2)$ is computed using the following formulas:

$$\begin{aligned} X_3 &= E \cdot F & \text{with } E &= (X_1 + Y_1) \cdot (X_2 + Y_2) - A - B & \text{with } A &= X_1 \cdot X_2 \\ Y_3 &= G \cdot H & F &= D - C & B &= Y_1 \cdot Y_2 \\ T_3 &= E \cdot H & G &= D + C & C &= 2{Z_1}^2 \\ Z_3 &= F \cdot G & H &= B - aA & D &= Z_1 \cdot Z_2 \end{aligned}$$

These formulas sum up to a computational complexity of 9M + 2D + 7add in $\mathbb{F}_p$ [17, p. 331].

The point doubling $(X_3, Y_3, T_3, Z_3) = 2 \cdot (X_1, Y_1, T_1, Z_1)$ is computed using the following formulas:

$$\begin{aligned} X_3 &= E \cdot F & \text{with } E &= (X_1 + X_1)^2 - A - B & \text{with } A &= {X_1}^2 \\ Y_3 &= G \cdot H & F &= G - C & B &= {Y_1}^2 \\ T_3 &= E \cdot H & G &= D + B & C &= 2{Z_1}^2 \\ Z_3 &= F \cdot G & H &= D - B & D &= aA \end{aligned}$$

These formulas sum up to a computational complexity of 4M + 4S + 1D + 7add in $\mathbb{F}_p$ [17, p. 333].

Comparing the addition formulas for projective twisted Edwards coordinates and extended twisted Edwards coordinates, we see a saving of 1M + 1S.

However, this is counteracted by the penalty for doubling in extended coordinates, which requires 1M more in extended coordinates compared to projective coordinates.

Since we converted the existing scalar multiplications algorithms available in RELIC to our twisted Edwards curve module, this penalty for doubling could easily be tested. Indeed it showed that, when using the same scalar multiplication algorithm, the extended coordinates were slightly slower than projective coordinates for twisted Edwards curves.

This lead us to implement an adjusted `LWNAF` algorithm specifically for extended twisted Edwards curve coordinates. Hisil *et al.* [17, p. 337] describe a mixed coordinate scalar multiplication algorithm which we adopt for the existing `LWNAF` implementation and describe in the next section in more detail.

4.3 Implementation of Mixed Coordinate Scalar Multiplication

We implemented the mixed coordinate scalar multiplication proposed by Hisil *et al.* [17] to take advantage of the extended twisted Edwards curve formulas. This algorithm will use the fact that one multiplication in the doubling formula is used for the T coordinate which is not a required input for the doubling formula. This means on subsequent doublings this multiplication can be skipped making the doubling formula of extended twisted Edwards coordinates as cheap as the formula for projective twisted Edwards coordinates.

The idea of the algorithm suggested by Hisil *et al.* follows the same idea as in [19]. During the scalar multiplication, the current value of the loop is not always held in extended coordinate representation but sometimes also in simple projective coordinate representation.

We adopted the existing `LWNAF` algorithm with the following two rules [17, p. 337] to the `LWNAF_MIXED` algorithm:

1. if a point doubling is followed by another doubling then skip the calculation of the T coordinate in the doubling
2. if a point doubling is followed by a point addition then use the full doubling formula plus extended coordinate addition

Listing 1 shows the relevant part of the adjusted `LWNAF` implementation.

```
ed_set_infty(r);
for (i = l - 1; i >= 0; i--, _k--) {
  n = *_k;
  if (n == 0) {
    /* doubling is followed by another doubling */
    if (i > 0) {
      ed_dbl_short(r, r);
    } else {
      /* use full extended coordinate doubling for last step */
      ed_dbl(r, r);
    }
  } else {
    ed_dbl(r, r);
    if (n > 0) {
      ed_add(r, r, t[n / 2]);
    } else if (n < 0) {
      ed_sub(r, r, t[-n / 2]);
    }
  }
}
```

Listing 1: Snippet from the `LWNAF_MIXED` algorithm implementation.

The basic `LWNAF` algorithm and the `LWNAF_MIXED` algorithm we have implemented are not hardened in any way against side-channel attacks. Okeya *et al.* [20] proposed modifications to the `LWNAF` algorithm which add protections against simple power analysis (SPA).

5 Evaluation

To test our implementation for correctness, the RELIC test suite for elliptic curves has been extended by a test suite for our twisted Edwards curve implementation, based on the existing tests for short Weierstrass curves over prime fields in RELIC. The test suite covers testing the addition formula for commutativity and associativity based on random points as input, the testing of the scalar multiplication methods in use with twisted Edwards curves and testing of utility functionality like conversion functions for elliptic curve points from and to a binary representation.

Our performance evaluation consists of three parts: evaluating the lower level performance of our additions to the RELIC elliptic curve cryptography (ECC) support by running the RELIC microbenchmark suite, a macrobenchmark running Elliptic Curve Diffie-Hellman (ECDH) shared secret calculation, and testing the higher level performance of an ECC-based identity-based signature (IBS).

We compare the existing Weierstrass curve implementation over $\mathbb{F}_p$, denoted as $E(\mathbb{F}_p)$, the new twisted Edwards curve implementation over $\mathbb{F}_p$ using projective coordinates, denoted as $\mathcal{E}(\mathbb{F}_p)$, and using extended coordinates, denoted as $\mathcal{E}^e(\mathbb{F}_p)$. This notation for $\mathcal{E}(\mathbb{F}_p)$ and $\mathcal{E}^e(\mathbb{F}_p)$ is the same as in [17, p. 337]. For the benchmarks using the short Weierstrass curve over $\mathbb{F}_p$, we use Curve25519 [8] which is birationally equivalent to the twisted Edwards curve ED25519. Currently, we have only implemented ED25519 in RELIC but more twisted Edwards curves can easily be added.

All three parts of our evaluation benchmarks cover all three elliptic curve configurations.

The benchmarks are executed on a high-end desktop platform (X86_64), a low power embedded platform (ARM11) and an Internet of Things (IoT) platform (ARM Cortex-M4). This covers complex instruction set computing (CISC) and reduced instruction set computing (RISC) platforms. The IoT platform is of particular interest due to its highly limited RAM compared to the other two platforms, and its missing CPU caches.

The detailed configurations of the test environments are shown in Table 4.

	PC	Pi	IoT
Device	Dell Optiplex 7010	Raspberry Pi	STM32F4discovery
Architecture	Intel	ARM	ARM
CPU	Corei5	ARM1176	ARM Cortex-M4
Word size	64 bit	32 bit	32 bit
Clock speed	2 GHz	800 MHz	168 MHz
L1 Cache	256 kB	32 kB	—
L2 Cache	1024 kB	(256 kB)	—
L3 Cache	6144 kB	—	—
RAM	16 GB	256 MB	192 kB
OS	Ubuntu 14.04	Debian 3.10.11-1+rpi7	RIOT [2][21]
Compiler	GCC 4.8.2	GCC 4.8.3	GCC 4.8.4

Note: The L2 Cache is used by the GPU on the Raspberry Pi and therefore is not available to the CPU.

Table 4: Test environments and compilers used for the benchmark.

Our benchmark procedure is generally the same for the low-level as for the high-level benchmark. We build RELIC, the newly introduced C++ wrapper for RELIC either and the benchmark code natively or cross-compiled.

All builds use the `-O2` optimization level. The `-O2` optimization level includes most optimizations provided by GCC. Using the highest optimization level `-O3` [3] would result in more aggressive unrolling of loops and further function inlining which increases the resulting code size. However, compact code is important for embedded IoT devices because memory is a critical resource.

Benchmark time is measured via low-level CPU cycle counters which are available on all of our test platforms. Each benchmark run executes the tested operation only one time. This is to prevent the cycle counter wrapping around its word size more than once and

[2]Commit `dc916ad4583def1af069e333affc28002380effe`

[3]See GCC documentation on optimization levels: `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`

thereby distorting our benchmark results.

Additionally, since the PC and PI test environments are multi-tasking systems, we do 30 benchmark runs and take the average cycle count for each benchmark as our final result. More details on the benchmark results, including minimum, maximum and spread of the values can be found in the appendix. On multi-tasking systems our benchmark code execution is subject to unavoidable side effects. One example of these side effects is preemptive context switching through disadvantage scheduler behavior during the benchmarks.

To further support reliable benchmark timings we apply the same guidelines as described by the SUPERCOP toolkit [4], i.e., disable hyper-threading where supported by the CPU, and disable energy saving features like Intel's TurboBoost which change the CPU clock speed on the fly depending on the current demands of the system.

On our IoT test platform, we only execute one run as RIOT supports cooperative scheduling resulting in a deterministic cycle count as benchmark result. For the benchmarks on the PC and Pi platforms, we execute 30 runs per platform and test. We use the mean over all runs of a test on a platform as final value for our tables for that test and platform. A statistical analysis for these benchmark runs can be found in the appendix.

The following three sections present the results and interpretation of the microbenchmark, the macrobenchmark and the benchmark of the IBS.

5.1 Elliptic Curve Microbenchmark

RELIC provides a microbenchmark that covers basic utility functions, arithmetic on elliptic curve points and various scalar multiplication algorithms. We used this benchmark to test the performance improvements of our implementations compared to the existing RELIC short Weierstrass curve implementation.

`add`, `sub`, `dbl` and `neg` describe the basic ECC group operations, i.e. point addition, point subtraction, point doubling and point negation, respectively. `mul` describes scalar multiplication of a point and `mul_gen` the scalar multiplication of the generator of the ECC group. For $E(\mathbb{F}_p)$ and $\mathcal{E}(\mathbb{F}_p)$ the scalar multiplication algorithm used is LWNAF. For $\mathcal{E}^e(\mathbb{F}_p)$ we used the newly implemented mixed coordinate multiplication LWNAF algorithm, the `LWNAF_MIXED` algorithm.

Table 5, Table 6 and Table 7 show that the current implementation for scalar multiplication of ECC points using extended twisted Edwards coordinates outperform the projective twisted Edwards coordinate formulas on all three systems in a similar manner.

However, on the ARM platforms the overall improvement of the twisted Edwards implementation compared to the short Weierstrass implementation is slightly smaller than on the Intel platform. On the ARM platforms the time required for scalar multiplication using twisted Edwards curves was cut down to roughly 76%-82% of the original short Weierstrass runtime. On the Intel platform the time was cut down to 72%-78% of the original short Weierstrass time.

The lower cycle count of the Pi system compared to IoT system can be explained in part by the level 1 data and instruction cache. The access time for the level 1 data cache is only

[4] See section "Reducing randomness in benchmarks" on `http://bench.cr.yp.to/supercop.html`

Benchmark	Short Weierstrass	Twisted Edwards		Twisted Edwards Extended	
	Cycles	Cycles	$\frac{\text{Twisted Ed.}}{\text{Weierstrass}}$	Cycles	$\frac{\text{Twisted Ed. Ext.}}{\text{Weierstrass}}$
`add`	10,072	7,952	0.79	6,762	0.67
`sub`	10,190	8,019	0.79	6,951	0.68
`dbl`	7,530	5,193	0.69	5,815	0.77
`neg`	274	264	0.96	303	1.11
`mul`	2,752,780	2,009,179	0.73	1,983,242	0.72
`mul_gen`	2,529,737	1,982,392	0.78	1,969,771	0.78

Table 5: Microbenchmark results from RELIC benchmark suite for PC system.

Benchmark	Short Weierstrass	Twisted Edwards		Twisted Edwards Extended	
	Cycles	Cycles	$\frac{\text{Twisted Ed.}}{\text{Weierstrass}}$	Cycles	$\frac{\text{Twisted Ed. Ext.}}{\text{Weierstrass}}$
`add`	58,102	45,712	0.79	38,859	0.67
`sub`	58,119	46,002	0.79	39,363	0.68
`dbl`	39,183	29,145	0.74	32,354	0.83
`neg`	441	325	0.74	504	1.14
`mul`	12,838,647	10,448,499	0.81	10,221,945	0.80
`mul_gen`	12,300,693	10,228,595	0.83	10,054,267	0.82

Table 6: Microbenchmark results from RELIC benchmark suite for Pi system.

3 cycles while an access to the primary memory can take up to 116 cycles [22]. The IoT system does not have any cache and every memory access is expensive.

In addition, it is clearly shown that the doubling for the extended projective coordinates is slower compared to simple projective coordinates for twisted Edwards curves. This confirms the relevance of a dedicated scalar multiplication algorithm for the extended coordinates, namely `LWNAF_MIXED`. This dedicated algorithm is used in our benchmarks for scalar multiplication for all twisted Edwards extended coordinate benchmarks.

5.2 Elliptic Curve Diffie–Hellman Macrobenchmark

We also conducted a macrobenchmark on the ECDH shared secret calculation. ECDH is one of the most popular applications of ECC and there are benchmark results of other groups to compare with, specifically the public eBACS [7] benchmark results.

Table 8 shows our best performance at 2.2 million cycles for an ECDH shared secret computation. The eBACS benchmark lists an ECDH shared secret computation for Curve25519 with 182,708 cycles, on a Intel Core i5 platform with 2.5 GHz compared to our 2 GHz

Benchmark	Short Weierstrass	Twisted Edwards		Twisted Edwards Extended	
	Cycles	Cycles	$\frac{\text{Twisted Ed.}}{\text{Weierstrass}}$	Cycles	$\frac{\text{Twisted Ed. Ext.}}{\text{Weierstrass}}$
`add`	544,140	416,110	0.76	352,330	0.65
`sub`	550,250	422,560	0.77	361,580	0.66
`dbl`	391,460	281,750	0.72	309,670	0.79
`neg`	4,560	3,640	0.80	5,750	1.26
`mul`	128,423,880	99,252,370	0.77	97,069,800	0.76
`mul_gen`	121,671,750	97,127,110	0.80	95,051,180	0.78

Table 7: Microbenchmark results from RELIC benchmark suite for IoT system.

Benchmark	Short Weierstrass	Twisted Edwards		Twisted Edwards Extended	
	Cycles	Cycles	$\frac{\text{Twisted Ed.}}{\text{Weierstrass}}$	Cycles	$\frac{\text{Twisted Ed. Ext.}}{\text{Weierstrass}}$
PC	2,867,657	2,238,981	0.78	2,219,457	0.77
Pi	13,141,632	11,001,186	0.84	10,877,226	0.83
IoT	133,496,430	107,280,770	0.80	105,243,430	0.79

Table 8: Microbenchmark results for ECDH shared secret benchmark

platform.

This huge difference is likely due to different factors:

- the implementation in eBACS uses a different scalar multiplication algorithm, an X-coordinate only Montgomery ladder. The Montgomery ladder has less computational complexity than the `LWNAF_MIXED` algorithm with extended twisted Edwards coordinates.
- there is difference in the benchmarking methods.
- our implementation of twisted Edwards curve support for the RELIC library is an early implementation without much effort spent on optimization.

5.3 vBNN-IBS Benchmark

In this section, we consider the impact of twisted Edwards curves on a higher level, specifically in the context of ID-based signature schemes. For this we compare signature generation and verification performance of ECC-based IBS on the same three hardware platforms used in the previous benchmarks. The IBS used for this benchmark is an implementation of vBNN-IBS [5] by Cao *et al.*

The benchmark procedure is the same as for the microbenchmark. 30 runs are made per benchmark configuration and the average cycle count is taken as result.

Benchmark	Short Weierstrass	Twisted Edwards		Twisted Edwards Extended	
	Cycles	Cycles	$\frac{\text{Twisted Ed.}}{\text{Weierstrass}}$	Cycles	$\frac{\text{Twisted Ed. Ext.}}{\text{Weierstrass}}$
key extraction	2,590,048	2,178,755	0.84	2,135,772	0.82
σ generation	2,593,784	2,317,637	0.89	2,266,173	0.87
σ verification	6,590,112	5,285,968	0.80	5,163,782	0.78

Table 9: vBNN-IBS results from RELIC benchmark suite for PC system.

Benchmark	Short Weierstrass	Twisted Edwards		Twisted Edwards Extended	
	Cycles	Cycles	$\frac{\text{Twisted Ed.}}{\text{Weierstrass}}$	Cycles	$\frac{\text{Twisted Ed. Ext.}}{\text{Weierstrass}}$
key extraction	12,487,573	10,683,095	0.86	10,530,146	0.84
σ generation	12,473,793	11,065,373	0.89	10,940,388	0.88
σ verification	30,291,546	25,269,266	0.83	24,903,077	0.82

Table 10: vBNN-IBS results from RELIC benchmark suite for Pi system.

Benchmark	Short Weierstrass	Twisted Edwards		Twisted Edwards Extended	
	Cycles	Cycles	$\frac{\text{Twisted Ed.}}{\text{Weierstrass}}$	Cycles	$\frac{\text{Twisted Ed. Ext.}}{\text{Weierstrass}}$
key extraction	122,511,580	101,872,350	0.83	100,265,910	0.82
σ generation	122,543,000	106,351,140	0.87	104,737,590	0.85
σ verification	389,525,330	312,752,570	0.80	307,320,620	0.79

Table 11: vBNN-IBS results from RELIC benchmark suite for IoT system.

Table 9 and Table 10 show an improvement down to about 85% of the original runtime for all operations of vBNN-IBS when used with twisted Edwards curves. Twisted Edwards curves with extended coordinates reduce the runtime even more but minimally.

For vBNN-IBS, the signature verification is the most expensive operation. All platforms show a speed-up down to between 80% and 83% of the original short Weierstrass curve runtime for twisted Edwards curves with simple projective coordinates over prime fields. The runtime for the twisted Edwards curves with extended coordinates ranges from 78% to 82% of the original short Weierstrass curve runtime.

Overall the improvements of our twisted Edwards curve implementation compared to the existing Weierstrass curve implementation in RELIC are in the same area across all three different benchmarks.

Of all three tested platforms the Pi platform shows the lowest relative speed-up in all three benchmarks. The CPU of the Pi platform, the PC platform and the IoT platform were released in 2003, 2009 and 2011, respectively. Considering that the CPU of the Pi is more than 5 years older than the other CPUs, it is technologically less advanced. This explains the low relative speed-up on the Pi platform.

5.4 Memory Footprint

The OS used on the IoT platform, RIOT, provides easy access to the maximal stack memory used by the program at runtime. Memory is a critical resource on constrained devices like our IoT test platform. The small available memory must be shared between our cryptographic functions and the rest of the application code. This means that security algorithms for constrained platforms need to have a small memory footprint. Otherwise nothing else but the security algorithm can run on the device and it would not be of any use.

Curve	Stack used (bytes)
Short Weierstrass	6,084
Twisted Edwards	5,828
Twisted Edwards Extended	6,156

Table 12: Memory usage on the IoT platform during the vBNN-IBS benchmark

Table 12 shows a reduced memory use of the twisted Edwards curve setting compared to classic short Weierstrass curves. This is likely due to the simplified addition formula which leads to simpler and shorter code. Note that the absolute values for stack usage are not representative for a realistic scenario since the benchmark covers running a key generation center (KGC) and two identities exchanging messages.

The increase in memory consumption for the twisted Edwards extended coordinate setting is explained by the additional coordinate per elliptic curve point during all computations and storage. The additional 4th T coordinate makes up for a 32 byte increase of memory usage per elliptic curve point in the code.

Twisted Edwards curves are in clear advantage over classic short Weierstrass curves here. twisted Edwards curves do not only come with a performance boost which is especially important in energy constrained environments, but also need less memory at runtime. This way there is more available runtime memory for the actual IoT application.

6 Conclusion

In this report, we presented the implementation of twisted Edwards elliptic curves for the RELIC library. We have evaluated its performance and the performance of the identity-based signature (IBS) vBNN-IBS on a variety of platforms including a highly constrained Internet of Things (IoT) board.

Implementing for and testing on constrained embedded hardware comes with additional challenges as keeping a balance between implementation convenience and memory use when the implementation language is C.

We showed that the improvements of twisted Edwards curves in RELIC support the use of newer asymmetric cryptography schemes in constrained environments. However compared to optimized and specialized implementations there is still a gap to close.

An outlook into possible further improvements to elliptic curve cryptography (ECC) performance in RELIC is twofold.

A possible area of improvement would profiling the code for heavily used code paths and analyzing this code for opportunities for inline assembler.

The adoption of further algorithmic improvements to the elliptic curve computations would be another area. This includes optimizations like the use of differential additions formulas [23] which further cut down the cost of the addition and doubling primitives for twisted Edwards curves.

The fastest Elliptic Curve Diffie-Hellman (ECDH) implementation in the eBATS [7] benchmark uses a twisted Edwards curve with a X-coordinate only Montgomery ladder [9] for scalar point multiplication. The X-coordinate only Montgomery ladder only requires a single value (X) in affine space or two values (X, Z) in projective space for the scalar multiplication. This reduces the required computational and storage complexity for point addition and doubling inside the multiplication loop. The use of the single coordinate Montgomery ladder is possible because every twisted Edwards curve is birationally equivalent to a Montgomery curve.

Symmetric cryptography already has common hardware acceleration available in modern desktop CPUs in form of the AES-NI instruction set. This support is available for years for the Intel/AMD 64-bit CPUs. The ARMv8-A is the first architecture that comes with similar hardware support for the ARM platform. However, it is a new architecture with first products released since 2013.

This shows that common hardware accelerated cryptography for constrained devices will not be available soon. Exceptions are custom developments which extend the basic CPU with external processing units optimized for the custom cryptographic needs. Using instruction set extensions and cryptographic coprocessors for heavily used finite field operations further performance and efficiency gains can be attained [24, 25].

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-key Cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, New York, NY, USA: ACM, Feb. 1978.

[2] D. F. Aranha and C. P. L. Gouvêa, *RELIC is an Efficient LIbrary for Cryptography*, `http://code.google.com/p/relic-toolkit/`.

[3] T. Markmann, “Performance Analysis of Identity-based Signatures,” Tech. Rep., 2014. [Online]. Available: `http://inet.cpt.haw-hamburg.de/teaching/ss-2014/master-projekt/tobias_markmann_prj1.pdf`.

[4] H. M. Edwards, “A normal form for elliptic curves,” *Bulletin of the American Mathematical Society*, vol. 44, no. 3, pp. 393–422, Providence, RI, USA: American Mathematical Society, 2007.

[5] X. Cao, W. Kou, L. Dang, and B. Zhao, “IMBAS: Identity-based multi-user broadcast authentication in wireless sensor networks,” *Computer Communications*, vol. 31, no. 4, pp. 659–667, Amsterdam, Netherlands: Elsevier, 2008.

[6] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, Florida, USA: CRC Press, 1996.

[7] D. J. Bernstein and T. Lange, Eds., *EBACS: ECRYPT Benchmarking of Cryptographic Systems*, accessed 3 February 2015, 2015. [Online]. Available: `http://bench.cr.yp.to`.

[8] D. J. Bernstein, “Curve25519: New Diffie-Hellman Speed Records,” in *Public Key Cryptography - PKC 2006*, ser. Lecture Notes in Computer Science, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958, Berlin, Heidelberg, Germany: Springer, 2006, pp. 207–228.

[9] P. L. Montgomery, “Speeding the Pollard and elliptic curve methods of factorization,” *Mathematics of Computation*, vol. 48, no. 177, pp. 243–264, Providence, RI, USA: American Mathematical Society, 1987.

[10] D. J. Bernstein and P. Schwabe, “NEON Crypto,” in *Cryptographic Hardware and Embedded Systems — CHES 2012*, ser. Lecture Notes in Computer Science, E. Prouff and P. Schaumont, Eds., vol. 7428, Berlin, Heidelberg, Germany: Springer-Verlag, 2012, pp. 320–339.

[11] R. de Clercq, L. Uhsadel, A. Van Herrewege, and I. Verbauwhede, “Ultra Low-Power Implementation of ECC on the ARM Cortex-M0+,” in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC ’14, San Francisco, CA, USA: ACM, 2014, pp. 1–6.

[12] J. López and R. Dahab, “High-Speed Software Multiplication in F_{2^m},” English, in *Progress in Cryptology — INDOCRYPT 2000*, ser. Lecture Notes in Computer Science, B. Roy and E. Okamoto, Eds., vol. 1977, Springer Berlin Heidelberg, 2000, pp. 203–212.

[13] D. J. Bernstein, T. Lange, and R. Rezaeian Farashahi, "Binary Edwards Curves," in *Cryptographic Hardware and Embedded Systems — CHES 2008*, ser. Lecture Notes in Computer Science, E. Oswald and P. Rohatgi, Eds., vol. 5154, Springer Berlin Heidelberg, 2008, pp. 244–265.

[14] D. J. Bernstein and T. Lange, "Faster Addition and Doubling on Elliptic Curves," in *Advances in Cryptology — ASIACRYPT 2007*, ser. Lecture Notes in Computer Science, K. Kurosawa, Ed., vol. 4833, Berlin, Heidelberg, Germany: Springer, 2007, pp. 29–50.

[15] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, ser. Discrete Mathematics and Its Applications. Abingdon, United Kingdom: Taylor & Francis, 2005.

[16] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards Curves," in *Progress in Cryptology — AFRICACRYPT 2008*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 5023, Berlin, Heidelberg, Germany: Springer, 2008, pp. 389–405.

[17] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson, "Twisted Edwards Curves Revisited," in *Advances in Cryptology — ASIACRYPT 2008*, ser. Lecture Notes in Computer Science, J. Pieprzyk, Ed., vol. 5350, Berlin, Heidelberg, Germany: Springer, 2008, pp. 326–343.

[18] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, Berlin, Heidelberg, Germany: Springer-Verlag, 2012.

[19] H. Cohen, A. Miyaji, and T. Ono, "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates," in *Advances in Cryptology — ASIACRYPT 1998*, ser. Lecture Notes in Computer Science, K. Ohta and D. Pei, Eds., vol. 1514, Berlin, Heidelberg, Germany: Springer, 1998, pp. 51–65.

[20] K. Okeya and T. Takagi, "The Width-w NAF Method Provides Small Memory and Fast Elliptic Scalar Multiplications Secure against Side Channel Attacks," in *Topics in Cryptology — CT-RSA 2003*, ser. Lecture Notes in Computer Science, M. Joye, Ed., vol. 2612, Berlin, Heidelberg, Germany: Springer, 2003, pp. 328–343.

[21] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proc. of the 32nd IEEE INFOCOM. Poster*, Piscataway, NJ, USA: IEEE Press, 2013.

[22] P. J. Drongowski, *Memory hierarchy and access time*, 2013. [Online]. Available: `http://sandsoftwaresound.net/raspberry-pi/raspberry-pi-gen-1/memory-hierarchy/`.

[23] L. Marin, "Differential Elliptic Point Addition in Twisted Edwards Curves," in *27th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Mar. 2013, pp. 1337–1342.

[24] A. Höller, N. Druml, C. Kreiner, C. Steger, and T. Felicijan, "Hardware/Software Co-Design of Elliptic-Curve Cryptography for Resource-Constrained Applications," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14, San Francisco, CA, USA: ACM, 2014, pp. 207–213.

[25] A. Targhetta, D. Owen, and P. Gratz, "The Design Space of Ultra-low Energy Asymmetric Cryptography," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Piscataway, NJ, USA: IEEE, Mar. 2014, pp. 55–65.

Appendices

Measurement Results

Benchmark	Minimum	Average	Median	Maximum	SD (σ)
Short Weierstrass					
add	9,914	10,072	9,957	11,461	329.77
sub	10,128	10,190	10,180.5	10,260	29.88
dbl	7,421	7,530	7,509	8,210	133.99
neg	271	274	273	279	1.83
mul	2,744,805	2,752,780	2,747,711.5	2,769,049	9,922.30
mul_gen	2,522,361	2,529,737	2,528,065	2,546,866	6,897.78
Twisted Edwards					
add	7,876	7,952	7,928.5	8,592	125.21
sub	7,942	8,019	8,011.5	8,137	42.30
dbl	5,142	5,193	5,178.5	5,255	33.47
neg	258	264	262	282	5.39
mul	2,002,409	2,009,179	2,008,183	2,029,378	5,014.11
mul_gen	1,975,802	1,982,392	1,979,839.5	2,002,221	7,149.99
Twisted Edwards Extended					
add	6,726	6,762	6,747	6,946	43.90
sub	6,913	6,951	6,945.5	7,084	31.88
dbl	5,763	5,815	5,813.5	5,917	30.56
neg	284	303	295.5	331	15.67
mul	1,979,384	1,983,242	1,982,975	2,002,056	3,921.36
mul_gen	1,963,029	1,969,771	1,965,886.5	1,982,699	8,481.99

Table 13: RELIC ECC microbenchmark result details for PC platform

Curve	Minimum	Average	Median	Maximum	SD (σ)
Short Weierstrass	2,858,625	2,867,657	2,863,809.5	2,892,076	9,152.88
Twisted Edwards	2,231,510	2,238,981	2,236,059.5	2,261,128	7,375.31
Twisted Edwards Extended	2,209,973	2,219,457	2,217,974	2,240,279	8,343.63

Table 14: RELIC ECDH microbenchmark result details for PC platform

Benchmark	Minimum	Average	Median	Maximum	SD (σ)
Short Weierstrass					
key extraction	2,570,981	2,590,048	2,586,961.5	2,624,979	12,739.45
σ generation	2,573,893	2,593,784	2,593,286	2,617,714	9,408.32
σ verification	6,457,207	6,590,112	6,597,665	6,708,968	58,185.01
Twisted Edwards					
key extraction	2,153,245	2,178,755	2,178,520	2,197,218	10,844.20
σ generation	2,295,408	2,317,637	2,316,813	2,353,461	13,364.52
σ verification	5,191,710	5,285,968	5,277,239.5	5,365,183	39,903.24
Twisted Edwards Extended					
key extraction	2,117,356	2,135,772	2,136,192.5	2,158,110	9,488.73
σ generation	2,245,462	2,266,173	2,264,449.5	2,302,786	12,780.58
σ verification	5,125,996	5,163,782	5,156,810	5,236,812	31,413.15

Table 15: vBNN-IBS benchmark result details for PC platform

Benchmark	Minimum	Average	Median	Maximum	SD (σ)
Short Weierstrass					
add	57,068	58,184	57,208.5	62,452	1,606.28
sub	58,055	58,673	58,134.5	65,192	1,448.97
dbl	39,158	39,587	39,220.5	45,821	1,231.07
neg	422	438	436	486	13.16
mul	12,817,426	12,857,835	12,840,762	12,917,203	31,440.31
mul_gen	12,243,690	12,270,165	12,263,303.5	12,312,997	21,611.93
Twisted Edwards					
add	45,676	46,049	45,716	47,508	646.19
sub	45,974	46,509	46,022	48,852	886.52
dbl	29,114	29,339	29,186.5	30,746	465.09
neg	321	327	325	373	9.10
mul	10,375,774	10,403,706	10,390,606.5	10,471,579	28,575.17
mul_gen	10,217,779	10,246,793	10,235,363	10,339,442	29,500.41
Twisted Edwards Extended					
add	38,815	39,200	38,917	40,882	602.20
sub	39,270	39,786	39,363.5	41,154	701.15
dbl	32,316	32,562	32,403.5	35,449	594.43
neg	500	514	506	569	17.66
mul	10,153,516	10,183,698	10,170,829	10,234,014	26,787.70
mul_gen	10,021,413	10,054,940	10,043,526	10,116,308	30,623.47

Table 16: RELIC ECC microbenchmark result details for Pi platform

Curve	Minimum	Average	Median	Maximum	SD (σ)
Short Weierstrass	13,090,396	13,141,632	13,121,832	13,205,319	39,565.74
Twisted Edwards	10,975,403	11,001,186	10,993,576	11,077,250	24,525.14
Twisted Edwards Extended	10,854,718	10,877,226	10,874,611	10,934,841	18,242.99

Table 17: RELIC ECDH microbenchmark result details for Pi platform

Benchmark	Minimum	Average	Median	Maximum	SD (σ)
Short Weierstrass					
key extraction	12,379,736	12,487,573	12,467,894	12,702,623	74,186.48
σ generation	12,377,568	12,473,793	12,455,732	12,648,606	66,124.68
σ verification	29,618,768	30,291,546	30,312,045	30,622,710	234,458.84
Twisted Edwards					
key extraction	10,600,637	10,683,095	10,682,977	10,826,918	51,435.21
σ generation	10,958,818	11,065,373	11,059,678	11,183,270	63,285.55
σ verification	24,976,280	25,269,266	25,192,384.5	25,760,335	190,828.01
Twisted Edwards Extended					
key extraction	10,404,731	10,530,146	10,522,690.5	10,678,465	64,064.06
σ generation	10,864,321	10,940,388	10,929,376.5	11,040,277	45,732.01
σ verification	24,449,496	24,903,077	24,897,385	25,176,369	187,517.30

Table 18: vBNN-IBS benchmark result details for Pi platform