Efficient Implementation of Pedersen Commitments Using Twisted Edwards Curves

Christian Franck and Johann Großschädl

University of Luxembourg,
Computer Science and Communications Research Unit,
6, Avenue de la Fonte, L–4364 Esch-sur-Alzette, Luxembourg
{christian.franck,johann.groszschaedl}@uni.lu

Abstract. Cryptographic commitment schemes are used in many contexts, whereby the size of the secret data and the security requirements depend on the target application. Using a software library that has been designed for other purposes (e.g., key-exchange or digital signatures) to compute commitments can be complicated or inefficient. We present in this paper a flexible implementation of Pedersen commitments based on elliptic curves in twisted Edwards form. The implementation supports a set of five curves of varying cryptographic strength, which are defined over 127, 159, 191, 223, and 255-bit pseudo-Mersenne prime fields. One can dynamically (i.e., at runtime) choose one of the curves according to the required level of security, and it is also possible to adapt to the size of the data to be committed by varying the number of base points. The point arithmetic is performed with optimized formulas using extended coordinates and dynamically pre-computed tables are utilized to speed up the scalar multiplication. Our implementation is written in ANSI C (with optional x86 assembler optimizations for the field arithmetic) and was compiled and tested successfully with Visual C on Windows, gcc on Linux, and clang on macOS. We present detailed benchmarking results for the field and point arithmetic on all five curves. When using an Intel Core i7 processor clocked at 2.7 GHz as test platform, we can compute more than 38,000 commitments per second on a twisted Edwards curve over a 127-bit field.

1 Introduction

Traditional coin-flipping, where Alice calls either ‘heads’ or ‘tails’ and then Bob flips a coin, is not secure when it is done online. The problem is that Alice does not see the coin toss and Bob could simply cheat and pretend the outcome was ‘tails’ if Alice has called ‘heads’. To prevent Bob from such cheating, Alice can compute a cryptographic commitment for either ‘heads’ or ‘tails’ and send it to Bob, instead of sending her choice directly. This commitment is binding Alice to her choice, but does not actually reveal it to Bob [9]. Alice will then only open the commitment and let Bob learn her choice once he has flipped the coin and announced the result. Opening the commitment discloses her choice and proves to Bob that it is authentic.

What differentiates cryptographic commitment schemes from other cryptographic primitives like digital signatures is that commitments often have to be secure only for a relatively short period of time. Taking the coin-flipping from above as example, the commitment needs to be secure for just the few seconds until Bob has flipped the coin. Therefore, the orders of common multiplicative groups and elliptic curves used by classical signature schemes (e.g., 256 bits in ECDSA [21]) are unnecessarily large for short-time commitments, which makes the computation of such commitments unnecessarily costly. Various short-lived applications could profit from more "lightweight" alternatives.

A well-known commitment scheme based on the complexity of the Discrete Logarithm Problem (DLP) was introduced by Pedersen in 1991 [22]. Pedersen commitments are especially interesting since they are computationally binding and unconditionally hiding. The latter means for the above example that even when Bob had unlimited computational power, he would not be able to obtain Alice's choice from the commitment [9]. On the other hand, the computational binding property implies that it is not possible for Alice to change her mind and open the commitment to a different choice, unless she has the ability to solve the DLP. Interestingly, it has also been shown that one can prolong the lifetime of a Pedersen commitment, which means that it is possible to securely replace a Pedersen commitment with a weak security parameter by a Pedersen commitment with a stronger security parameter if the need arises [10]. Hence, one can start with lightweight commitments and switch to stronger ones later on.

In this paper, we introduce a software library that was specifically designed for the computation of Pedersen commitments [22, 5] so that they can easily be adapted to meet different requirements. The library features:

- *Adjustable security level (commitment size).* The library can be configured to use elliptic curves of different strength (i.e., different cardinality) in steps of 32 bits, ranging from 127 to 255 bits. In this way, the library is capable to generate commitments of different length and can be adapted for various security requirements.
- *Adaptable to the size of the secret data.* In order to generate commitments for secret data of "large" size, the library supports extended Pedersen commitments [5] on elliptic curves, which require a multi-scalar multiplication $C = s_1P_1 + s_2P_2 + \cdots + rQ$ with an arbitrary number of base points. This allows to maintain the homomorphic properties of Pedersen commitments even if the size of the secret data is large.
- *State-of-the-art elliptic curves.* A collection of five elliptic curves in twisted Edwards form [3], defined over 127, 159, 191, 223, and 255-bit pseudo-Mersenne prime fields, comes with the library. These curves satisfy all common security and efficiency requirements. In particular, the parameter a of these curves is -1, which facilitates the use Hisil et al's optimized point addition formulas for extended coordinates introduced in [16].
- *Fast fixed-base scalar multiplication with pre-computation.* The library uses the fixed-base windowing method described in [4], which employs pre-computed tables containing multiples of the base point in order to speed up the

scalar multiplication. It is possible to dynamically pre-compute such tables for any number of base points, and also the number of table entries can be adapted to allow for trade-offs between speed and RAM footprint.
- *Pre-computation of affine or extended affine coordinates.* One can choose to pre-compute either two or three coordinates per point in the table, and the corresponding optimized formulas will be used for the point addition. The two-coordinate variant is a bit slower, but reduces the table size by 33%.
- *Generic C and optimized x86 assembler code.* For maximal compatibility the library was written in ANSI C99, and performance-critical field-arithmetic operations were additionally implemented in x86 assembler to minimize the execution time. The source code was successfully compiled (and tested) on three different operating systems using three different compilers.
- *Resistance against timing attacks.* All arithmetic operations (with only one exception, namely inversion in the prime field) as well as the table look-ups have constant (i.e. operand-independent) execution time. The field inversion adopts the extended Euclidean algorithm [15] in combination with a simple multiplicative masking technique to thwart timing attacks.

The rest of the paper is organized as follows. In Sect. 2, we review the basics of Pedersen commitments and elliptic curve cryptography. Then, in Sect. 3, we motivate and describe the details of our implementation. In Sect. 4, we present benchmarking results for some selected curves, and in Sect. 5, we discuss some possible applications of our library. We conclude with remarks in Sect. 6.

2 Background

2.1 Commitment Schemes

Cryptographic commitment schemes allow one to commit to some secret value without having to reveal it (at the time of making the commitment), but it is possible to reveal the value later and to prove that the revealed value is indeed the correct value [9]. In general, a commitment protocol is performed between a committer and a verifier, and consists of the two following steps:

1. *Commit:* In order to commit to a secret value s, the committer chooses a random value r and sends the commitment
$$C = \text{commit}(s, r)$$
to the verifier. The knowledge of C does not provide the verifier with any information about the secret value s.
2. *Open:* In order to open the commitment, the committer sends (s, r) to the verifier. The committer is bound to the value s, which means it is hard to create another pair of values (s', r') such that
$$C = \text{commit}(s', r').$$

As explained in [9], a commitment scheme can be either computationally binding and unconditionally hiding, or it can be unconditionally binding and computationally hiding. However, it can never be unconditionally hiding and unconditionally binding at the same time.

Pedersen Commitments. The Pedersen commitment scheme, in its original form as described in [22], uses a prime-order subgroup of $\mathbb{Z}_p^*$ as basic algebraic structure. However, it is also possible to embed the computation of Pedersen commitments into an elliptic-curve group $E(\mathbb{F}_p)$. Assume $E(\mathbb{F}_p)$ contains two points P and $Q = \alpha P$, both having prime order q, whereby the scalar $\alpha < q$ is unknown. In this setting, the commitments are of the form

$$C = \text{commit}(s, r) = sP + rQ.$$

Similar to "classical" Pedersen commitments operating in $\mathbb{Z}_p^*$, the elliptic-curve variants are *unconditionally hiding* since every possible value of the secret s is equally likely to be committed in C. More precisely, for any $s' \neq s$, there exists an $r' \neq r$ such that $\text{commit}(s', r') = C = \text{commit}(s, r)$; this r' can be obtained by computing $r' = (s - s')/\alpha + r \bmod q$. Therefore, the verifier can not learn anything about s from C, even if she had unlimited computing power. Furthermore, elliptic-curve Pedersen commitments are *computationally binding* since the committer can not open a commitment to s as $s' \neq s$, unless she is able to solve the Elliptic Curve Discrete Logarithm Problem (ECDLP). Namely, if the committer could find a pair (s', r') that commits to the same C as (s, r), then it would be easy for her to get $\alpha = (s - s')/(r' - r) \bmod q$, which contradicts the hardness assumption for the ECDLP.

The idea of Pedersen commitments can also be extended to multiple values $s_1, \ldots, s_n$, as shown in [5]. The commitments are then of the form

$$C = s_1 P_1 + s_2 P_2 + \cdots + s_n P_n + rQ. \tag{1}$$

As a consequence, one can use Pedersen commitments to commit to messages of arbitrary length. We assume the points $P_1, P_2, \ldots, Q$ are chosen at random and their respective discrete logarithms are unknown.

Pedersen commitments also have homomorphic properties. Given the commitments

$$C_1 = s_1 P + r_1 Q \text{ and } C_2 = s_2 P + r_2 Q$$

for the values s_1 and s_2, one can compute a commitment C_{12} corresponding to the secret value $s_1 + s_2$ and the random value $r_1 + r_2$, with

$$C_{12} = (s_1 + s_2)P + (r_1 + r_2)Q = (s_1 P + r_1 Q) + (s_2 P + r_2 Q) = C_1 + C_2.$$

In order to preserve this homomorphic property for longer messages, one can not just use a hash function $\mathcal{H}$ and compute $C = \mathcal{H}(s_1 | \cdots | s_n)P + rQ$, but it is necessary to compute a commitment as in Eq. (1).

2.2 Twisted Edwards Curves

Twisted Edwards (TE) curves were introduced in 2008 by Bernstein et al [3] as a generalization of Edwards curves [11]. Formally, a TE curve is defined by an equation of the form

$$E : ax^2 + y^2 = 1 + dx^2y^2 \tag{2}$$

over a non-binary finite field $\mathbb{F}_q$, where a and d are distinct, non-zero elements of $\mathbb{F}_q$. Every TE curve is birationally-equivalent over $\mathbb{F}_q$ to a Montgomery curve and, thus, also to an elliptic curve in Weierstraß form. The order of a TE curve is divisible by 4 (i.e. TE curves have a co-factor of $h \geq 4$), and every TE curve contains a point of order 2, namely $(0, -1)$. Given two points $P_1 \in E(\mathbb{F}_q)$ and $P_2 \in E(\mathbb{F}_q)$, their sum $P_3 = P_1 + P_2$ can be computed as

$$(x_3, y_3) = (x_1, y_1) + (x_2, y_2) = \Big(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}\Big).$$

The point $\mathcal{O} = (0, 1)$ serves as neutral element of the addition, and the negative of a point (x_1, y_1) is $(-x_1, y_1)$. Note that the addition formula specified above is *unified*, which means it can also be used for point doubling. Furthermore, as shown in [3], the given addition formula is *complete* (i.e. yields the correct sum for any pair of points, including corner cases like $P_1 = \mathcal{O}$, $P_2 = \mathcal{O}$, $P_2 = -P_1$) when the curve parameter a is a square and d a non-square in $\mathbb{F}_q$. In order to avoid costly inversions in the point arithmetic, one normally uses a projective coordinate system. A well-known example are the so-called extended projective coordinates from [16], which allow for particularly efficient point addition when $a = -1$. A point in extended project coordinates is represented by a quadruple $(X : Y : T : Z)$ where $T = XY/Z$. The projective curve equation is

$$(aX^2 + Y^2)Z^2 = Z^4 + dX^2Y^2, \tag{3}$$

which can be simplified to $aX^2 + Y^2 = Z^2 + dT^2$. The TE curves we use for the implementation of Pedersen commitments feature a fast and complete addition law, meaning that $a = -1$ is a square in the underlying prime field.

3 Implementation Details

In this section, we give an overview of our software implementation of Pedersen commitments using TE curves. We aimed to reach three main goals, namely (i) high performance, (ii) high scalability, and (iii) support for a wide range of x86 platforms. In order to achieve fast execution times, we decided to implement all performance-critical operations, in particular the multiplication and squaring in the underlying field, not just in C but also in Assembly language. Our software is scalable because it supports Pedersen commitments of varying cryptographic "strength" (using TE curves of different order) without the need to recompile the source code. Finally, to support many platforms, we developed our software for the standard x86 architecture and refrained from using 64-bit instructions

or SIMD extensions such as SSE. In this way, our implementation of Pedersen commitments can run on a plethora of x86-compatible platforms, ranging from high-end 64-bit Intel Core processors down to embedded 32-bit variants like the Intel Quark [18] for systems on chip. As part of our future research, we plan to extend the software with an optimized 64-bit Assembler implementation of the field arithmetic so that it can reach peak performance on 64-bit processors.

3.1 Prime-Field Arithmetic

The arithmetic operations in the underlying field, especially multiplication and squaring, have a massive impact on the overall execution time of elliptic curve cryptosystems, and our TE-based Pedersen commitments are no exception. It is common practice to adopt finite fields defined by primes of a "special" form in order to maximize the efficiency of the modular reduction operation. As will be explained in Subsect. 3.4, the TE curves we use are defined over fields based on *pseudo-Mersenne primes*, which are primes of the form $p = 2^k - c$, where c is small in relation to 2^k [15]. The elements of $\mathbb{F}_p$ are integers with a length of up to k bits, and the product z of two such integers a, b is at most $2k$ bits long. To reduce z modulo p, one can exploit $2^k \equiv c \bmod p$, which results in a reduction technique with linear complexity. More concretely, in order to reduce z modulo p, the product z is first split up into an upper part z_H and a lower part z_L so that $z = z_H 2^k + z_L$; then, z_H is multiplied by c and $z_H c$ is added to z_L. These steps are repeated with the obtained result and, finally, subtractions of p have to be performed to get a fully reduced result (see e.g. [20] for details).

As will be specified in full detail in Subsect. 3.4, the prime fields we use are defined by pseudo-Mersenne primes of the form $p = 2^k - c$ where k is a multiple of 32 minus 1 (e.g. $k = 255$) and c is at most 29 bits long (i.e. c fits in a single x86 register). We represent the field elements by arrays of 32-bit words of type `uint32_t`, which means in the case of $k = 255$ that an array has eight words. In the beginning of this section we mentioned already that our software contains two implementations of the field arithmetic, one written in C and the other in x86 Assembly language. The C implementation is generic in the sense that the arithmetic functions can process operands of any length. Every function of the C arithmetic library gets besides the arrays for the operands and the result an extra parameter that specifies the number of words the arrays consist of. On the other hand, the x86 Assembler library comes with a dedicated implementation for each supported operand length, which means, for example, that it contains five functions for modular multiplication, optimized for 127, 159, 191, 223, and 255-bit fields. Each of these functions was carefully hand-tuned and loops were fully unrolled to maximize performance.

A multiplication in a pseudo-Mersenne prime field is normally performed in two steps: first, the field elements are multiplied, yielding a double-length product, and thereafter a modular reduction is carried out, taking into account the special form of the prime. Our Assembler implementation applies the so-called *product-scanning method*, which means the 64-bit word-products resulting from multiplying pairs of 32-bit words are summed up in a column-wise fashion (see

[15, Algorithm 2.10] and [20, Algorithm 1] for a more formal description). The biggest challenge one has to tackle when implementing the this technique on an x86 processor is the small register file, consisting of just eight general-purpose registers, one of which is the stack pointer register `ESP`. The `MUL` instruction in x86 reads one operand from the `EAX` register, while the second operand can be either in a register or in memory. It executes an unsigned 32-bit multiplication and places the 64-bit product in the `EDX:EAX` register pair. Our implementation of the modular multiplication stores the double-length product of the two field elements in a temporary array on the stack, which is accessed via `ESP`. Of the remaining seven registers, three hold the column sum, two contain the pointers to the operand arrays, and `EAX`/`EDX` are used to execute `MUL` instructions.

The square of a field element can be computed using fewer `MUL` instructions than the product of two distinct field elements. Our implementation is based on the optimized squaring technique described in [20].

3.2 Point Arithmetic

The to date most efficient way of performing point arithmetic on a TE curve is to use the extended coordinates proposed by Hişil et al [16]. In this coordinate system, a point $P = (x, y)$ is represented by a quadruple $(X : Y : T : Z)$ where $x = X/Z$, $y = Y/Z$, $xy = T/Z$, and $Z \neq 0$. Such extended coordinates can be seen as homogenous projective coordinates of the form $(X : Y : Z)$, augmented by a fourth coordinate $T = XY/Z$ that corresponds to the product xy in affine coordinates. The neutral element $\mathcal{O}$ is given by $(0 : 1 : 0 : 1)$, and the negative of a point in extended coordinates is $(-X : Y : -T : Z)$. A point represented in standard affine coordinates as (x, y) can be converted to extended coordinates by simply setting $X = x$, $Y = y$, $T = xy$, and $Z = 1$. The re-conversion is done in the same way as for homogenous projective coordinates through calculation of $x = X/Z$ and $y = Y/Z$, which costs an inversion in the underlying field.

In the following, we roughly explain the unified addition/doubling formulae using extended coordinates as given by Hişil et al in [16, Sect. 3.1]. Let P_1 and P_2 be two arbitrary points on a TE curve represented in extended coordinates of the form $(X_1 : Y_1 : T_1 : Z_1)$ and $(X_2 : Y_2 : T_2 : Z_2)$ where $Z_1, Z_2 \neq 0$. When $a = -1$ (as is the case for all our curves from Subsect. 3.4), a unified addition $P_3 = P_1 + P_2 = (X_3 : Y_3 : T_3 : Z_3)$ consists of the following operations.

$$\begin{gathered} A \leftarrow (Y_1 - X_1) \cdot (Y_2 - X_2), \quad B \leftarrow (Y_1 + X_1) \cdot (Y_2 + X_2), \quad C \leftarrow k \cdot T_1 \cdot T_2, \\ D \leftarrow 2Z1 \cdot Z_2, \quad E \leftarrow B - A, \quad F \leftarrow D - C, \quad G \leftarrow D + C, \quad H \leftarrow B + A, \\ X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \quad T_3 \leftarrow E \cdot H, \quad Z_3 \leftarrow F \cdot G \end{gathered} \tag{4}$$

The factor k used in the computation of C is $-2d/a$, which means in our case $k = 2d$ since the parameter $a = -1$ for all our TE curves. It is easy to observe that computational cost of the point addition amounts to nine multiplications (9M) in the underlying prime field, plus a few "cheaper" field operations like additions. When P_2 is given in affine coordinates (i.e. $Z_2 = 1$), the addition is a so-called "mixed addition" and requires only eight multiplications (8M). Hişil

et al also introduced a formula for doubling a point $P_1 = (X_1 : Y_1 : T_1 : Z_1)$ so that the result $P_3 = 2P_1$ is also given in extended coordinates. For TE curves with parameter $a = -1$, the sequence of operations to double a point is

$$\begin{gathered} A \leftarrow X_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow 2Z_1^2, \quad D \leftarrow -A, \quad E \leftarrow (X_1 + Y_1)^2 - A - B, \\ G \leftarrow D + B, \quad F \leftarrow G - C, \quad H \leftarrow D - B, \\ X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \quad T_3 \leftarrow E \cdot H, \quad Z_3 \leftarrow F \cdot G. \end{gathered} \tag{5}$$

A doubling carried out via Eq. (5) requires four multiplications (4M) as well as four squarings (4S) in the underlying field. Unlike the addition formula given in Eq. (4), the doubling operation does not use the auxiliary coordinate T_1 of the point P_1. Therefore, the computation of T_3 in Eq. (5) could be simply omitted whenever a doubling is followed by another doubling. A similar observation can be made for the addition because the coordinate T_3 in Eq. (4) does not need to be computed when the subsequent operation is a point doubling.

In order to accelerate the point doubling operation, we do not compute the auxiliary coordinate T_3 as in Eq. (5) but output the two factors E and H it is composed of instead. In this way, the resulting point $P_3 = 2P_1$ consists of five coordinates instead of four, which means P_3 is actually represented in the form of a quintuple $(X_3 : Y_3 : E_3 : H_3 : Z_3)$. The coordinate T_3 is split up into factors E_3 and H_3 such that $E_3H_3 = T_3 = X_3Y_3/Z_3$, thereby saving a multiplication in the point doubling. The subsequently-executed operation can recover T_3, when needed, by simply multiplying E_3 by H_3. Of course, such a modification of the doubling requires to adapt the point addition accordingly [8]. We modified the addition formula specified in Eq. (4) to output the two factors $E = B - A$ and $H = B + A$ instead of $T_3 = EH$. In this case, when the addition is performed with P_1 represented by $(X_1 : Y_1 : E_1 : H_1 : Z_1)$ as input, the auxiliary coordinate $T_1 = E_1H_1$ has to be computed first since it is used as operand. However, this modification has no impact on the overall cost of the point addition since the computation of the coordinate $T_3 = EH$ is simply replaced by computing the coordinate $T_1 = E_1H_1$. On the other hand, the cost of the point doubling gets reduced from 4M + 4S to 3M + 4S thanks to this optimization.

Our software for Pedersen commitments actually computes so-called mixed additions, which means point P_1 is given in projective coordinates (in our case extended projective coordinates in the form of a quintuple, see above), whereas P_2 is represented using affine coordinates. We implemented two variants of the mixed addition; the first expects P_2 in standard affine (x, y) coordinates, while in the other variant, P_2 must be provided in extended affine coordinates of the form (u, v, w) where $u = (x - y)/2$, $v = (x + y)/2$, and $w = dxy$, similar to the mixed addition in e.g. [4] and [20]. The exact formula for the former variant is specified in Appx. A (Algorithm 1) and has a cost of 8M plus a multiplication by the parameter d, which is fast for all our TE curves since d is small. On the other hand, the latter variant takes only 7M (because the product dxy is pre-computed), but this performance gain comes at the expense of requiring three coordinates for P_2, which can be undesirable on certain platforms or for certain scalar multiplication techniques that pre-compute and store many points.

3.3 Computation of Pedersen Commitments

The high-level strategy we use to compute the commitments is a generalization of the fast fixed-base exponentiation techniques described in [6, 19, 23], which is also used by Bernstein et al in [4]. The basic idea is that a scalar multiplication with a fixed base point, i.e. an operation of the form $S = \kappa P$ where P is known a priori (e.g. it is the generator of an elliptic-curve subgroup), can be computed faster using a (possibly small) table with pre-computed points. To avoid cache attacks, we use constant-time table look-ups as described in e.g. [4]. In order to accelerate the scalar multiplication, we write the k-bit scalar κ as

$$\kappa = \sum_{i=0}^{\lceil k/4 \rceil} \kappa_i \cdot 16^i$$

with $0 \leq \kappa_i \leq 15$ for $i \in \{0, 1, \ldots, \lceil k/4 \rceil\}$ and pre-compute 15 multiples of the base point P, namely the set $\{16^i P, (2 \cdot 16^i)P, \ldots, (15 \cdot 16^i)P\}$ for every i from 0 to $\lceil k/4 \rceil$. This reduces the computation of S to $\lceil k/4 \rceil$ point additions since

$$S = \sum_{i=0}^{\lceil k/4 \rceil} (\kappa_i \cdot 16^i)P.$$

As will be detailed in Subsect. 3.4, we use primes of the form $p = 2^k - c$, where k is a multiple of 32 minus 1, and also the bitlength of our scalars is a multiple of 32 minus 1, similar to [2]. Thus, we can assume $0 \leq \kappa_{\lceil k/4 \rceil} \leq 7$, which means we can also utilize signed coefficients κ'_i with $-8 \leq \kappa'_i < 8$, so that

$$S = \sum_{i=0}^{\lceil k/4 \rceil} (\kappa'_i \cdot 16^i)P.$$

Signed coefficients have the big advantage that only eight pre-computed points (namely $\{16^i P, (2 \cdot 16^i)P, \ldots, (8 \cdot 16^i)P\}$) are needed for every i. They have no impact on performance because the negative of a point (x, y) on a TE curve is simply $(-x, y)$. To further reduce the number of pre-computed points, we write

$$S = \sum_{i=0}^{\lceil k/8 \rceil} (\kappa'_{2i} \cdot 16^{2i})P + 16 \cdot \sum_{i=0}^{\lceil k/8 \rceil} (\kappa'_{2i+1} \cdot 16^{2i})P.$$

Hence, at the cost of four point doublings (since $16 = 2^4 = 2 \cdot 2 \cdot 2 \cdot 2$), we can halve the number of necessary pre-computed points, and the pre-computations need to be done only for $i \in \{0, \ldots, \lceil k/8 \rceil\}$ instead of $i \in \{0, \ldots, \lceil k/4 \rceil\}$. In the context of Pedersen commitments, we can have expressions of the form

$$C = s_1 P_1 + s_2 P_2 + \cdots + s_n P_n + rQ,$$

which have several base points. We "integrate" these computations according to

$$C = \sum_{i=0}^{\lceil k/8 \rceil} \kappa_{2i}^{(1)} 16^{2i} P_1 + \cdots + r_{2i} 16^{2i} Q + 16 \sum_{i=0}^{\lceil k/8 \rceil} \kappa_{2i+1}^{(1)} 16^{2i} P_1 + \cdots + r_{2i+1} 16^{2i} Q^{16}$$

so that we have to do the four point doublings of the second term only once. As the number of base points gets larger, the relative cost of these four doublings decreases, and it can make sense to do some further modifications to obtain an expression of the form

$$C = \sum_{i=0}^{\lceil k/16 \rceil} \cdots + 16 \left(\sum_{i=0}^{\lceil k/16 \rceil} \cdots + 16 \left(\sum_{i=0}^{\lceil k/16 \rceil} \cdots + 16 \sum_{i=0}^{\lceil k/16 \rceil} \cdots \right) \right),$$

which again reduces the number of pre-computed points at the expense of eight point doublings (i.e. we have to perform 12 doublings altogether).

Our software for Pedersen commitments supports all these possibilities; it is up to the user to choose which trade-off between the amount of pre-computed points and number of point doublings suits best for a certain application. The amount of memory m (in bytes) required to store the pre-computed points can be calculated using the formula

$$m = b \cdot \frac{(k+1)^2}{4(t+1)},$$

where b is the number of base points, $t \in \{2, 4, 8, 16, 32\}$ is the number of times that the four point doublings are performed, and k denotes the bitlength of the scalar (which is, in our case, the same as the bitlength of the underlying prime field and is always a multiple of 32 minus 1). The factor $(k+1)^2$ in the above formula implies that choosing a smaller prime p whenever possible will reduce the memory requirements significantly.

The high-level API of our x86 software for the computation and verification of Pedersen commitments supports the following six functions:

- *Pre-computation:* $\Omega = \text{precomp}(\Gamma, (P_1, P_2, \ldots), \tau)$. Pre-computes points to speed up a fixed-base scalar multiplication using the TE curve parameters $\Gamma = (k, c, d)$, a set of base points $(P_1, P_2, \ldots)$, and a parameter τ to trade memory usage for speed.
- *Commitment:* $C = \text{commit}(\Gamma, (s_1, \ldots, s_n), r, \Omega)$. Computes a commitment for the set of secret values $(s_1, \ldots, s_n)$ using the random number r and the pre-computed points Ω. The output is compressed as described in [4].
- *Verification:* $\{0, 1\} = \text{verify}(C, \Gamma, (s_1, \ldots, s_n), r, \Omega)$. Verifies whether the set $(s_1, \ldots, s_n)$ and number r correspond to the commitment C.
- *Compression:* $C = \text{compress}(\Gamma, A)$. Converts the point A in standard affine to a commitment C (in compressed representation).
- *Decompression:* $A = \text{decompress}(\Gamma, C)$. Decompresses a commitment C to recover the x and y coordinate of the corresponding affine point A.
- *Addition:* $A = \text{add}(\Gamma, A_1, A_2)$. Adds the two affine points A_1 and A_2.

3.4 Supported TE Curves

A TE curve needs to meet several security and efficiency criteria in order to be suitable for cryptography applications. A discussion of these criteria is outside

the scope of this paper; we refer the interested reader to [14] and the references given therein. Our software for Pedersen commitments comes with a set of five TE curves, which are defined over pseudo-Mersenne prime fields having lengths of 127, 159, 191, 223, and 255 bits. The co-factor of the curves is $h = 8$, and so they provide security levels of 62, 78, 94, 110, and 126 bits. Concretely, the five TE curves we use are specified by the following equations.

$$-x^2 + y^2 = 1 + 182146x^2y^2 \bmod 2^{127} - 507$$
$$-x^2 + y^2 = 1 + 49445x^2y^2 \bmod 2^{159} - 91$$
$$-x^2 + y^2 = 1 + 141087x^2y^2 \bmod 2^{191} - 19$$
$$-x^2 + y^2 = 1 + 987514x^2y^2 \bmod 2^{223} - 235$$
$$-x^2 + y^2 = 1 + 4998299x^2y^2 \bmod 2^{255} - 19$$

The latter four curves are taken from [14], where it is described how they were generated and what security properties they meet. We generated the first curve (i.e. the curve based on the 127-bit field) from scratch, following the guidelines in [14]. However, it must be noted that solving the ECDLP in a 124-bit elliptic curve subgroup is well within reach for a well-funded adversary; therefore, this curve is only suitable for commitments with short-time security requirements in the area of a few seconds. Also the adequacy of the curve over the 159-bit field (providing a security level of roughly 78 bits) must be carefully evaluated. The main characteristics of all five curves are summarized in Table 2.

Our software is not restricted to these curves and can be easily extended to support other pseudo-Mersenne prime fields and TE curves, provided that the following conditions are fulfilled. First, the constant c of the pseudo-Mersenne prime $p = 2^k - c$ is at most 29 bits long and k is a multiple of 32 minus 1. The resulting prime p must be congruent to 5 modulo 8 so that $a = -1$ is a square in $\mathbb{F}_p$ (and the TE addition law can be complete [3]) and square roots modulo p (which are needed for the decompression of compressed curve points [4]) can be computed efficiently via Atkin's method [1]. Second, the parameter d of the TE curve is at most 32 bits long and a is fixed to -1 so that the fast addition formula proposed by Hişil et al [16] can be used. The resulting TE curve needs to have a co-factor of $h = 8$ and meet all other requirements listed in [14].

4 Benchmarking Results

In this section, we present some benchmarks for the field arithmetic operations and the computation of commitments for five different security levels using the curves given above. We made an effort to ensure the C and Assembler source codes can be compiled (and execute correctly) with three different compilers on three different operating systems, namely Microsoft Visual C on Window 7, gcc on Linux, and clang on macOS. All timings were collected with a test program that was compiled with clang version 3.9.0 (using -O2 optimization level) and executed on an Intel Core i7 CPU clocked at 2.7 GHz. We measured the cycle counts of the different operations following the approach described in [17].

Table 1. Computation time of field operations on a 2.7 GHz Core i7 CPU.

Prime p	$2^{127} - 507$	$2^{159} - 91$	$2^{191} - 19$	$2^{223} - 235$	$2^{255} - 19$
C99					
multiplication	95 cycles	123 cycles	165 cycles	200 cycles	256 cycles
squaring	89 cycles	120 cycles	156 cycles	191 cycles	230 cycles
Assembler					
multiplication	51 cycles	68 cycles	85 cycles	111 cycles	140 cycles
squaring	48 cycles	59 cycles	71 cycles	87 cycles	108 cycles

4.1 Field Operations

As explained in Subsect. 3.1, our software contains two implementations of the field arithmetic: one is speed-optimized (i.e. written in x86 Assembly language) and supports 127, 159, 191, 223, and 255-bit primes, whereas the second aims for high flexibility and is "generic" so that it can be used for pseudo-Mersenne primes of arbitrary length (in steps of 32 bits). This second implementation is written in ANSI C99 and not particularly optimized in any way. Table 1 shows the cycle counts of multiplication and squaring (including modular reduction) on an Intel Core i7 processor. We can observe that the field operations become significantly more expensive as the bitlength of the prime increases. For example, multiplication and squaring for 255-bit operands is roughly 2.5 times more costly as the same operations for operands of a length of 127 bits. Squaring is about 23% faster than multiplication (for 255-bit operands), but the difference decreases for shorter operands or when the operations are written in C. While the assembler implementations of multiplication are nearly two times as fast as their C counterparts, the speed-up factor due to Assembly programming grows even above two for squaring.

The performance of the field arithmetic on an Intel Core processor could be much improved by using 64-bit instructions or the SSE extensions. However, as stated in the previous section, we aimed to support a wide range of x86 platforms, and hence we restricted ourselves to the standard 32-bit x86 instruction set. In this way, the software can also run on embedded x86 processors like the Intel Quark [18], which features neither 64-bit instructions nor SSE.

4.2 Commitments

As mentioned in Subsect. 3.3, we use tables with pre-computed points to speed up the computation of the Pedersen commitments. These tables are generated dynamically for the chosen TE curve, taking into account the number of base points. It is possible to trade performance for RAM requirements by choosing between standard affine and extended affine coordinates, and by increasing the number of point doublings as described in Subsect. 3.3.

The results in Table 2 show that the size of the pre-computed tables grows rapidly as the elliptic-curve groups (and underlying fields) become larger. When

Table 2. Parameters and benchmark results for selected twisted Edwards curves on a 2.7 GHz Core i7 CPU.

Prime p	$2^{127}-507$	$2^{191}-19$	$2^{223}-235$	$2^{255}-19$
Curve par. d	182146	141087	987514	4998299
Security (Safe curves)				
subgr. order n	2^{124}	2^{188}	2^{220}	2^{252}
pollard-rho	$2^{61.83}$	$2^{93.83}$	$2^{109.83}$	$2^{125.83}$
embed. degree	$n/4$	n	$n/2$	$n/9$
tr. Frobenius	$-2^{63.17}$	$-2^{96.00}$	$-2^{112.19}$	$-2^{127.40}$
CM field discr.	$-2^{126.75}$	$-2^{189.99}$	$-2^{221.46}$	$-2^{254.64}$
twist secure	yes	yes	yes	yes
rigid design	yes	yes	yes	yes
Size				
commit. size	128 bit	192 bit	224 bit	256 bit
Precomputed Affine coordinates				
Simple commitment with 2 base points				
computation	71158 cycles	152229 cycles	204995 cycles	253231 cycles
precomp.	3892328 cycles	10658120 cycles	14830488 cycles	21045653 cycles
table size	8192 bytes	18432 bytes	25088 bytes	32768 bytes
Multiple commitment with 10 base points				
computation	307827 cycles	676167 cycles	928502 cycles	1117830 cycles
precomp.	19696098 cycles	50817550 cycles	74457348 cycles	104493723 cycles
table size	40960 bytes	92160 bytes	125440 bytes	163840 bytes
Multiple commitment with 25 base points				
computation	758717 cycles	1651434 cycles	2284449 cycles	2797713 cycles
precomp.	49074094 cycles	127289535 cycles	186253419 cycles	260234216 cycles
table size	102400 bytes	230400 bytes	313600 bytes	409600 bytes
Precomputed Extended Affine coordinates				
Simple commitment with 2 base points				
computation	71912 cycles	150958 cycles	204109 cycles	235712 cycles
precomp.	3949639 cycles	10222641 cycles	14974472 cycles	21170345 cycles
table size	12288 bytes	27648 bytes	37632 bytes	49152 bytes
Multiple commitment with 10 base points				
computation	311964 cycles	670083 cycles	923140 cycles	1099020 cycles
precomp.	19833071 cycles	51258665 cycles	74953927 cycles	107634723 cycles
table size	61440 bytes	138240 bytes	188160 bytes	245760 bytes
Multiple commitment with 25 base points				
computation	774411 cycles	1651590 cycles	2257933 cycles	2633260 cycles
precomp.	49852794 cycles	128884006 cycles	188669593 cycles	263449068 cycles
table size	153600 bytes	345600 bytes	470400 bytes	614400 bytes

the size of the underlying field doubles from 127 to 255 bits, the table size and computation time increases by a factor of roughly between three and four. The time spent for the pre-computation of tables grows by even larger factors. This confirms that committing to a secret value through two 128-bit commitments is much cheaper in terms of table size and computation time than using a single 256-bit commitment instead. However, as stated in Subsect. 3.4, commitments generated using a TE curve of such small order can only be considered secure for a very short period of time (e.g. a few seconds)

We can further see in Table 2 that for the TE curve over the 255-bit prime field, the computation of a commitment using pre-computed points in extended affine coordinates is only marginally faster than when using conventional affine coordinates. However, the tables holding points in extended affine coordinates are 50% larger than the tables containing conventional affine coordinates. It is remarkable that the advantage of pre-computing three coordinates vanishes the smaller the order of the curve becomes. For example, for the TE curve over the 127-bit field, the variant using standard affine coordinates turns out to be even faster than the approach based on extended affine coordinates.

The verification of a commitment consists in using the revealed value(s) to accomplish the same computations that were made when the commitment was created. Therefore, the computational cost of a verification is the same as the cost of computing a commitment.

5 Applications

Since Pedersen commitments are used in an increasing number of contexts, we believe that our software can be useful in many application domains. To give concrete examples, we discuss potential usage scenarios in two areas.

First, we look at the field of untraceable communication where two variants of the dining cryptographers protocol [7] have been introduced that both make extensive use of Pedersen commitments. The dining cryptographers protocol is multiparty protocol in which all participants first establish pairwise secret keys and then later they publish random-looking values derived from said keys. The sum of all the published values can reveal the message, but it is impossible to determine which participant was the sender. For many years, this protocol was considered to be impractical because a malicious participant could disrupt the communication by publishing wrong values and remain undetected. In the two more recent approaches [12, 13], this problem has been tackled using Pederson commitments. During the initialization phase, the n participants are required to compute n or n^2 Pedersen commitments for each subsequent transmission round. This makes a large number of commitments to pre-compute during the initialization and many commitments to verify in each transmission round. In settings where the expected lifetime of commitments is in the range of seconds to minutes, an elliptic curve of small order can be used, e.g. our curve over the 127-bit field). The number of bases depends on the size of the messages; when only short signalling messages are transmitted, a few bases will suffice.

Another field in which Pedersen commitments can be used is the long-time archiving of digital documents containing sensitive data [10]. Certain countries like Estonia require hospitals to store large amounts of medical data, and there are also lots of sensible government data that has to be kept secure for several decades. To guarantee the privacy and authenticity of the data without having to reveal it, one may opt to generate Pedersen commitments. In this case, the expected lifetime and the required security level of the commitments has to be much higher than in the previous example, and so one may decide to utilize an elliptic curve providing a security level of 128 bits (e.g. our TE curve over the 255-bit field) or even above. The data can be longer, so one might chose to go for a larger number of bases. It is possible to replace a commitment after some time by an equivalent commitment with stronger security parameters [10].

These two examples clearly illustrate that different usage scenarios require different kinds of Pedersen commitments. While in the first scenario there is a need for "lightweight" commitments for small messages, the second scenario is about long-term security for large(r) documents. Our software was designed in such a way that it can easily be configured for any of these use cases.

6 Concluding Remarks

We presented an x86 software library specifically aimed at computing Pedersen commitment based on TE curves with optimized formulae for the addition and doubling of points. The arithmetic functions in the underlying pseudo-Mersenne prime fields have been implemented in both ANSI C and x86 Assembly. On the higher level it is possible to dynamically pre-compute points for fast fixed-base scalar multiplication with a variable number of base points.

The results of the benchmark tests confirm that the stronger commitments based on large-order curves are much more expensive in terms of computation time and memory requirements than their more "lightweight" counterparts. To provide a concrete example, a 256-bit commitment can be three to four times more expensive than a 128-bit commitment. It makes therefore sense to have a software that allows to adjust the commitments to their expected lifetime, the size of the secret data, and the available memory for pre-computed points.

Finally, we discussed possible application scenarios for the software, but we believe there are many more. We hope that the software will prove to be useful to researchers who plan to implement protocols using Pedersen commitments.

References

1. A. O. Atkin. Probabilistic primality testing (summary by F. Morain). In *INRIA Research Report 1779*, pp. 159–163. INRIA, 1992. Available for download at `http://algo.inria.fr/seminars/sem91-92/atkin.pdf`.
2. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *Public Key Cryptography — PKC 2006*, vol. 3958 of *Lecture Notes in Computer Science*, pp. 207–228. Springer Verlag, 2006.

3. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In *Progress in Cryptology — AFRICACRYPT 2008*, vol. 5023 of *Lecture Notes in Computer Science*, pp. 389–405. Springer Verlag, 2008.
4. D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, Sept. 2012.
5. S. Brands. Rapid demonstration of linear relations connected by Boolean operators. In *Advances in Cryptology — EUROCRYPT '97*, vol. 1233 of *Lecture Notes in Computer Science*, pp. 318–333. Springer Verlag, 1997.
6. E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation (Extended abstract). In *Advances in Cryptology — EUROCRYPT '92*, vol. 658 of *Lecture Notes in Computer Science*, pp. 200–207. Springer Verlag, 1992.
7. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, Jan. 1988.
8. D. Chu, J. Großschädl, Z. Liu, V. Müller, and Y. Zhang. Twisted Edwards-form elliptic curve cryptography for 8-bit AVR-based sensor nodes. In *Proceedings of the 1st ACM Workshop on Asia Public-Key Cryptography (AsiaPKC 2013)*, pp. 39–44. ACM Press, 2013.
9. I. B. Damgård. Commitment schemes and zero-knowledge protocols. In *Lectures on Data Security: Modern Cryptology in Theory and Practice*, vol. 1561 of *Lecture Notes in Computer Science*, chapter 3, pp. 63–86. Springer Verlag, 1999.
10. D. Demirel and J. Lancrenon. How to securely prolong the computational bindingness of Pedersen commitments. Cryptology ePrint Archive, Report 2015/584, 2015. Available for download at `http://eprint.iacr.org/2015/584`.
11. H. M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, July 2007.
12. C. Franck and U. K. Sorger. Untraceable VoIP communication based on DC-nets. *CoRR*, abs/1610.06549, 2016. Available for download at `http://arxiv.org/abs/1610.06549`.
13. C. Franck and J. van de Graaf. Dining cryptographers are practical (Preliminary version). *CoRR*, abs/1402.2269, 2014. Available for download at `http://arxiv.org/abs/1402.2269`.
14. S. Ghatpande, J. Großschädl, and Z. Liu. A family of lightweight twisted Edwards curves for the Internet of things. Preprint, submitted for publication, 2017.
15. D. R. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Verlag, 2004.
16. H. Hişil, K. K.-H. Wong, G. Carter, and E. Dawson. Twisted Edwards curves revisited. In *Advances in Cryptology — ASIACRYPT 2008*, vol. 5350 of *Lecture Notes in Computer Science*, pp. 326–343. Springer Verlag, 2008.
17. Intel Corporation. How to Benchmark Code Execution Times on Intel® IA-32 and IA-64 Instruction Set Architectures. White paper, available for download at `http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf`, 2010.
18. Intel Corporation. Intel® Quark™ SoC X1000. Product specification, available online at `http://ark.intel.com/products/79084/Intel-Quark-SoC-X1000-16K-Cache-400-MHz`, 2015.
19. C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology — CRYPTO '94*, vol. 839 of *Lecture Notes in Computer Science*, pp. 95–107. Springer Verlag, 1994.

20. Z. Liu, J. Großschädl, L. Li, and Q. Xu. Energy-efficient elliptic curve cryptography for MSP430-based wireless sensor nodes. In *Information Security and Privacy — ACISP 2016*, vol. 9722 of *Lecture Notes in Computer Science*, pp. 94–112. Springer Verlag, 2016.
21. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). FIPS Publication 186-4, available for download at `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`, July 2013.
22. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO '91*, vol. 576 of *Lecture Notes in Computer Science*, pp. 129–140. Springer Verlag, 1992.
23. N. Pippenger. On the evaluation of powers and related problems (Preliminary version). In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science (FOCS 1976)*, pp. 258–263. IEEE Computer Society Press, 1976.

A Algorithms for Point Arithmetic

Algorithm 1. Point addition on a twisted Edwards curve with $a = -1$

Input: Point P_1 in extended projective coordinates $(X_1 : Y_1 : E_1 : H_1 : Z_1)$ satisfying $E_1H_1 = T_1 = X_1Y_1/Z_1$, point P_2 in affine coordinates (x_2, y_2), curve parameter d
Output: Sum $P_3 = P_1 + P_2$ in extended projective coordinates $(X_3 : Y_3 : E_3 : H_3 : Z_3)$

1: $T_1 \leftarrow E_1 \cdot H_1$
2: $E_3 \leftarrow Y_1 - X_1$
3: $H_3 \leftarrow Y_1 + X_1$
4: $U_2 \leftarrow y_2 - x_2$
5: $V_2 \leftarrow y_2 + x_2$
6: $X_3 \leftarrow E_3 \cdot U_2$
7: $Y_3 \leftarrow H_3 \cdot V_2$
8: $E_3 \leftarrow Y_3 - X_3$
9: $H_3 \leftarrow Y_3 + X_3$
10: $U_2 \leftarrow x_2 \cdot y_2$
11: $V_2 \leftarrow 2d \cdot U_2$
12: $X_3 \leftarrow T_1 \cdot V_2$
13: $Y_3 \leftarrow 2Z_1$
14: $U_2 \leftarrow Y_3 - X_3$
15: $V_2 \leftarrow Y_3 + X_3$
16: $X_3 \leftarrow E_3 \cdot U_2$
17: $Y_3 \leftarrow V_2 \cdot H_3$
18: $Z_3 \leftarrow U_2 \cdot V_2$
19: **return** $(X_3 : Y_3 : E_3 : H_3 : Z_3)$

Algorithm 2. Point doubling on a twisted Edwards curve with $a = -1$

Input: Point P_1 in extended projective coordinates $(X_1 : Y_1 : E_1 : H_1 : Z_1)$ satisfying $E_1H_1 = T_1 = X_1Y_1/Z_1$
Output: Double $P_3 = 2 \cdot P_1$ in extended projective coordinates $(X_3 : Y_3 : E_3 : H_3 : Z_3)$

1: $E_3 \leftarrow {X_1}^2$
2: $H_3 \leftarrow {Y_1}^2$
3: $T_1 \leftarrow E_3 - H_3$
4: $H_3 \leftarrow E_3 + H_3$
5: $X_3 \leftarrow X_1 + Y_1$
6: $E_3 \leftarrow {X_3}^2$
7: $E_3 \leftarrow H_3 - E_3$
8: $Y_3 \leftarrow {Z_1}^2$
9: $Y_3 \leftarrow 2Y_3$
10: $Y_3 \leftarrow T_1 + Y_3$
11: $X_3 \leftarrow E_3 \cdot Y_3$
12: $Z_3 \leftarrow Y_3 \cdot T_1$
13: $Y_3 \leftarrow T_1 \cdot H_3$
14: **return** $(X_3 : Y_3 : E_3 : H_3 : Z_3)$