Compressed Pairings

Michael Scott\textsuperscript{1,*} and Paulo S. L. M. Barreto\textsuperscript{2}

\textsuperscript{1} School of Computing, Dublin City University.
Ballymun, Dublin 9, Ireland.
mike@computing.dcu.ie

\textsuperscript{2} Escola Politécnica, Universidade de São Paulo.
Av. Prof. Luciano Gualberto, tr. 3, 158.
BR 05508-900, São Paulo(SP), Brazil.
pbarreto@larc.usp.br

Abstract. Pairing-based cryptosystems rely on bilinear non-degenerate maps called pairings, such as the Tate and Weil pairings defined over certain elliptic curve groups. In this paper we show how to compress pairing values, how to couple this technique with that of point compression, and how to benefit from the compressed representation to speed up exponentiations involving pairing values, as required in many pairing based protocols.

Keywords: pairing-based cryptosystem, efficient implementation.

1 Introduction

With the discovery of a viable identity-based encryption scheme based on the Weil pairing [5], pairing-based cryptography has become of great interest to cryptographers. Since then, pairing-based protocols – many with novel properties – have been proposed for key exchange [30], digital signature [6], encryption [5], and signcryption [28]. Although the Weil pairing was initially proposed as a suitable construct for the realisation of such protocols, it is now usually accepted that the Tate pairing is preferable for its greater efficiency. Supersingular elliptic curves were originally proposed as a suitable setting for pairing-based schemes; recent work has shown that certain ordinary curves are equally suitable, and offer greater flexibility in the choice of security parameters [3,26]. Fast computer algorithms for the computation of the Tate pairing on both supersingular and ordinary curves have been suggested in [1,3,12].

The Tate pairing calculation involves an application of Miller’s algorithm [24] coupled to a final exponentiation to get a unique value. A typical protocol step requires the calculation of a pairing value followed by a further exponentiation of the result.

In this paper we explore the concept of \textit{compressed pairings}, their efficient computation, and the subsequent processing (typically exponentiation) of pairing values. Our main contribution is to show that one can effectively reduce the

\textsuperscript{*} Supported in part by Enterprise Ireland RIF grant IF/2002/0312/N
bandwidth occupied by pairing values without impairing security nor processing time; in some cases, one even obtains a 30%-40% speed enhancement. Our work gives further motivation for the approach of Galbraith et al. [14], who investigate the bit security of pairing values and show that taking the trace causes no loss of security.

This paper is organized as follows. Section 2 introduces basic mathematical concepts. Section 3 discusses laddering exponentiation of pairing values, and introduces a laddering variant of the BKLS [1] algorithm to compute pairings. Section 4 describes how to compress pairing values to half length, and establishes a connection with the techniques of point compression and point reduction. Section 5 defines a ternary exponentiation ladder for finite fields in characteristic 3. Section 6 describes how to compress pairing values to one third of their length, presents a more efficient and slightly simpler version of the Duursma-Lee algorithm [11] that enables pairing computation in compressed form, and discusses improved variants of point compression and point reduction in characteristic 3. We summarise our work in section 7.

2 Mathematical Preliminaries

The theory behind elliptic curve cryptography is well documented in standard texts. The reader is referred to [23] for more background.

Let $p$ be a prime number, $m$ a positive integer and $\mathbb{F}_p^m$ the finite field with $p^m$ elements; $p$ is said to be the characteristic of $\mathbb{F}_p$, and $m$ is its extension degree. Unless otherwise stated, we assume $p \neq 2$ throughout this paper.

Let $q = p^m$. An elliptic curve $E(\mathbb{F}_q)$ is the set of solutions $(x, y)$ over $\mathbb{F}_q$ to an equation of form $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, where $a_i \in \mathbb{F}_q$, together with an additional point at infinity, denoted $O$. The same equation defines curves over $\mathbb{F}_{q^k}$ for $k > 0$ (although note that the $a_i$ remain in $\mathbb{F}_q$). The number of points on an elliptic curve $E(\mathbb{F}_{q^k})$, denoted $\#E(\mathbb{F}_{q^k})$, is called the order of the curve over the field $\mathbb{F}_{q^k}$.

An (additive) Abelian group structure is defined on $E$ by the well known secant-and-tangent method [29]. Let $n = \#E(\mathbb{F}_{q^k})$. The order of a point $P \in E$ is the least nonzero integer $r$ such that $rP = O$, where $rP$ is the sum of $r$ terms equal to $P$. The order of a point divides the curve order. For a given integer $r$, the set of all points $P \in E$ such that $rP = O$ is denoted $E[r]$. We say that $E[r]$ has embedding degree $k$ if $r | q^k - 1$ and $r \not| q^s - 1$ for any $0 < s < k$. In this paper we assume $k > 1$. It is in fact not difficult to find suitable curves with this property for relatively small values of $k$ as described in [2, 7, 10]. We are interested here in curves where $k$ is even, as this case facilitates fast calculation of the Tate pairing [3].

For our purposes, a divisor is a formal sum $A = \sum_P a_P(P)$ of points on the curve $E(\mathbb{F}_{q^k})$. An Abelian group structure is defined on the set of divisors by the addition of corresponding coefficients in their formal sums; in particular, $nA = \sum_P (n a_P)(P)$. The degree of a divisor $A$ is the sum $\deg(A) = \sum_P a_P$. Let $f : E(\mathbb{F}_{q^k}) \to \mathbb{F}_{q^k}$ be a function on the curve and let $\deg(f) = 0$. We define
\( f(A) = \prod_P f(P)^{a_P}. \) The divisor of a function \( f \) is any \( f = \sum_P \text{ord}_P(f)(P). \) A divisor \( A \) is called principal if \( A = (f) \) for some function \( f \). A divisor \( A \) is principal if and only if \( \text{deg}(A) = 0 \) and \( \sum_P a_P P = O \) [23, theorem 2.25]. Two divisors \( A \) and \( B \) are equivalent, \( A \sim B \), if their difference \( A - B \) is a principal divisor. Let \( P \in E(F_q)[r] \) where \( r \) is coprime to \( q \), and let \( \mathcal{A}_P \) be a divisor equivalent to \( (P - (O)) \); under these circumstances the divisor \( r \mathcal{A}_P \) is principal, and hence there is a function \( f_P \) such that \( (f_P) = r \mathcal{A}_P = r(P) - r(O) \). The (reduced) Tate pairing of order \( r \) is the map \( e_r : E(F_q)[r] \times E(F_{q^r}) \to \mathbb{F}_{q^r}^* \) given by \( e_r(P, Q) = f_P(D)^{(q^r - 1)/r} \) for some divisor \( D \sim (Q) - (O) \). The Tate pairing is bilinear and non-degenerate; assuming \( k > 1 \), one gets \( e_r(P, Q) \neq 1 \) if \( Q \) is chosen from a coset containing a point of order \( r \) which is linearly independent from \( P \). The computation of \( f_P(D) \) is achieved by an application of Miller’s algorithm [24], whose output is only defined up to an \( r \)-th power in \( \mathbb{F}_{q^r}^* \). The final exponentiation to the power of \( (q^r - 1)/r \) is needed to produce a unique result, and it also makes it possible to compute \( f_P(Q) \) rather than \( f_P(D) \) [1]. Sometimes we will drop the \( r \) subscript of the Tate pairing, writing simply \( e(P, Q) \).

### 2.1 Lucas sequences

Lucas sequences provide a relatively cheap way of implementing \( \mathbb{F}_{q^2} \) exponentiation in a subgroup whose order divides \( q + 1 \). They have been extensively studied in the literature, and a fast “ladder” algorithm for their computation has been developed [18,19,32], using ideas originally developed by Lehmer and Montgomery [20,27]. Lucas sequences have been suggested as a suitable vehicle for certain public-key schemes (see [4]). The laddering algorithm can in fact be used as an alternative to the standard square-and-multiply approach to exponentiation in any Abelian group, but it is particularly well-suited for Lucas sequences and certain parameterisations of elliptic curves [19]. The authors of [19] go on to emphasise that the laddering algorithm requires very little memory, facilitates parallel computing, and has a natural resistance to side-channel attacks when used in a cryptographic context.

The Lucas sequence consists of a pair of functions \( U_k, V_k : \mathbb{F}_q \times \mathbb{F}_q \to \mathbb{F}_q \). Commonly one is interested in computing \( U_k(P, 1) \) and \( V_k(P, 1) \) for some field element \( P \), in which case we write simply \( U_k(P) \) and \( V_k(P) \) or omit the arguments altogether. For this distinguished case the sequences are defined as

\[
U_0 = 0, \quad U_1 = 1, \quad U_{k+1} = PU_k - U_{k-1} \\
V_0 = 2, \quad V_1 = P, \quad V_{k+1} = PV_k - V_{k-1}
\]

Only the \( V_k \) sequence needs to be explicitly evaluated, as we also have the relationship

\[
U_k = (PV_k - 2V_{k-1})/(P^2 - 4)
\]

The fast laddering algorithm is described in Appendix A. Lucas sequences are useful in the exponentiation of certain field elements, as we will see next.
3 Exponentiating pairing values

We consider first the case of embedding degree \( k = 2 \) (although the following discussion also covers the case \( k = 2d \) with the substitution \( q \to q^d \)). Recall that we assume the characteristic to be odd.

We represent an element of the field \( \mathbb{F}_{q^2} \) as \( x + iy \), where \( x, y \in \mathbb{F}_q \), and \( i^2 = \delta \) for some quadratic non-residue \( \delta \in \mathbb{F}_q \). Assume in what follows that all arithmetic is in the field \( \mathbb{F}_q \).

The final exponentiation in this case consists of a raising to the power of \( (q - 1)(q + 1)/r \). This can be considered in two parts - exponentiation to the power of \( q - 1 \) followed by exponentiation to the power of \( (q + 1)/r \). Now if the output of Miller’s algorithm is \( x + iy \in \mathbb{F}_{q^2} \), then

\[
(x + iy)^{q^{-1}} = (x + iy)^q/(x + iy) = (x - iy)/(x + iy)
\]

which is obviously much quicker than the standard square-and-multiply algorithm. The element \( a + ib \equiv (x + iy)^{q^{-1}} \) calculated in this fashion has the property:

\[
a^2 - \delta b^2 = 1
\]

where \( a^2 - \delta b^2 \) is called the norm of \( a + ib \); this property, easily verified by simple substitution, is maintained under any subsequent exponentiation. An element of this form in \( \mathbb{F}_{q^2} \) is called unitary [16]. Also observe that \( (a + ib)^{-1} = (a - ib) \) for a unitary element. In fact, any element of \( \mathbb{F}_{q^2} \) whose order divides \( q + 1 \) will have this property.

A unitary element can obviously be determined up to the sign of \( b \) from \( a \) alone, using equation 1. And this is our first observation - the output of the Tate algorithm contains some considerable redundancy. It could be represented by a single element of \( \mathbb{F}_q \) and a single bit to represent the sign of \( b \), rather than as a full element of \( \mathbb{F}_{q^2} \).

One can efficiently raise a unitary element of \( \mathbb{F}_{q^2} \) to a power \( m \) by means of Lucas sequences. This is a consequence of the observation that

\[
(a + bi)^m = V_m(2a)/2 + U_m(2a)bi,
\]

as one can verify by induction. As pointed out above, only \( V_m(2a) \) needs to be explicitly calculated.

If \( M \) is a multiplication and \( S \) a squaring in \( \mathbb{F}_q \), then the computational cost of this method to compute \( (a + bi)^m \) is therefore \( 1M + 1S \) per step, where a step involves the processing associated with a single bit of \( m \) (see appendix A).

The conventional binary exponentiation algorithm in \( \mathbb{F}_{q^2} \) takes 1 squaring and about 1/2 multiplication in \( \mathbb{F}_{q^2} \) for an overall cost of roughly \( 2S + 5M/2 \) per step. If \( \delta = -1 \), then this can be reduced to \( 2S + 3M/2 \) per step. Thus the improved algorithm costs about 60% as much as the basic binary square-and-multiply method. When memory is not an issue the binary algorithm can be

---

\( ^3 \) If \( a + bi \) is unitary and \( \delta = -1 \), one can compute \( (a + bi)^2 \) as \( (2a^2 - 1) + [(a + b)^2 - 1]i \), and \( (a + bi)(c + di) \) as \( (u - v) + (w - u - v)i \) where \( u = ac, v = bd, w = (a + b)(c + d) \).
implemented by using windowing techniques, as described in [15]. However the lathering algorithm proposed here for unitary elements will always be faster than a conventional binary algorithm for a general element in $\mathbb{F}_q^*$. Note that this improvement is relevant not only for the second part of the final exponentiation of the Tate pairing, but for any exponentiation directly involving pairing values, as happens in many pairing-based protocols [5,17,28].

3.1 A lathering pairing algorithm

For $U, V \in E(\mathbb{F}_q)$, define $g_{U,V}$ to be the line through $U$ and $V$. For all $a, b \in \mathbb{Z}$, the line function satisfies $(g_{ap,bP}) = (aP) + (bP) + (-|a + b|P) - 3(O)$.

Let $P \in E(\mathbb{F}_q)$, and for $c \in \mathbb{Z}$ let $f_c$ be a function with divisor $(f_c) = c(P) − (cP) − (c − 1)(O)$. One can show that $f_{a+b}(D) = f_a(D) \cdot f_b(D) \cdot g_{ap,bp}(D)/g_[a+b]p_{-|a+b|}p(D)$ up to a constant nonzero factor. This is called Miller’s formula. In the computation of the Tate pairing $e_r(P, Q)$ for even $k$ and a careful choice of $P$ and $Q$ (see [1,3]), this formula can be simplified to $f_{a+b}(Q) = f_a(Q) \cdot f_b(Q) \cdot g_{ap,bp}(Q)$.

Let $(r_0, \ldots, r_0)_2$ be the binary representation of $r$. By coupling Miller’s simplified formula with Montgomery’s scalar multiplication ladder, we get a lathering version of the BKLS algorithm [1] to compute $e_r(P, Q)$:

Lathering BKLS algorithm to compute $e_r(P, Q)$:

- $v_0 \leftarrow 1$, $v_1 \leftarrow 1$
- $R_0 \leftarrow P$, $R_1 \leftarrow 2P$
- for $i \leftarrow t - 1$ downto 0 do
  - if $r_i = 0$ then
    - $v_0 \leftarrow v_0^2 \cdot g_{R_0,R_0}(Q)$, $R_1 \leftarrow R_0 + R_1$
    - $R_0 \leftarrow 2R_0$, $v_1 \leftarrow v_0 \cdot g_{R_0,P}(Q)$
  - else
    - $v_1 \leftarrow v_0^2 \cdot g_{R_1,R_1}(Q)$, $R_0 \leftarrow R_0 + R_1$
    - $R_1 \leftarrow 2R_1$, $v_0 \leftarrow v_1 \cdot g_{R_1,-P}(Q)$
  - end if
- end for
- return $v_0^{(q^k-1)/r}$

Although this algorithm has no computational advantage over the original BKLS, it may be useful in the same context of the lathering algorithms described in [19].

4 Compressing pairings to half length

Instead of keeping the full $a + bi$ value of the Tate pairing, it may be possible for cryptographic purposes to discard $b$ altogether, leaving the values defined
only up to conjugation, which means one of the pairing arguments will only be
defined up to a sign:

\[ e(P, Q) = a + bi \Rightarrow a - bi = (a + bi)^{-1} = e(P, Q)^{-1} = e(P, -Q). \]

This is similar to the point reduction technique, whereby instead of keeping
\( Q = (x, y) \) one only keeps the abscissa \( x \).

**Definition 1.** The \( \mathbb{F}_q \)-trace of an element \( u \in \mathbb{F}_{q^2} \) is the sum of the conjugates
of \( u \), \( \text{tr}(u) = u + \bar{u}^q \).

Notice that \( \text{tr}(a + ib) = (a + ib) + (a - ib) = 2a \), in effect discarding the
imaginary part. We define the **compressed Tate pairing** \( e(P, Q) \) as \( \text{tr}(e(P, Q))^4 \).

4.1 Point reduction

Point reduction is an optimization technique introduced by Miller in 1985 [25]. It
consists of basing cryptographic protocols solely on the \( x \) coordinate of the points
involved rather than using both coordinates. This setting is possible because the
\( x \) coordinate of any multiple of a given point \( P \) depends only on the \( x \) coordinate
of \( P \). A related but less efficient technique is that of point compression, which
consists of keeping not only the \( x \) coordinate but also a single bit \( \beta \) from the \( y \)
coordinate to choose between the two roots \( y_\pm = \pm \sqrt{a^3 + ax + b} \).

Some pairing-based cryptosystems have been originally defined to take profit
from point reduction. An example is the BLS signature scheme [6], where the
signature of a message represented by a curve point \( M \) under the signing key \( s \) is
the \( x \) coordinate \( \sigma \) of the point \( S = sM \). This means that, implicitly, the actual
signature is \( \pm S \) rather than \( S \) alone. To verify a BLS signature, the verifier checks
whether \( e(M, V) = e(\pm S, Q) \), where the verification key is \( V = sQ \). Incidentally,
the verification key itself can be reduced to its \( x \) coordinate (say, \( \xi \)), even though
this possibility does not seem to have been considered by the authors of BLS.

4.2 Coupling point reduction with compressed pairings

Verifying a BLS signature involves computing a point \( V \in \{ V, -V \} \) from \( \xi \),
a point \( S' \in \{ S, -S \} \) from \( \sigma \) and checking whether \( e_r(M, V') = e(S', Q) \) or
\( e(M, V') = e(S', Q)^{-1} \). Using the property that any pairing value \( z \) is unitary
(and hence \( z^{-1} = \overline{z} \)), one can simply check whether \( \text{tr}(e(M, V')) = \text{tr}(e(S', Q)) \).
This is especially interesting, since a compressed pairing \( e(P, Q) \) is precisely
\( \text{tr}(e(\pm P, \pm Q)) \).

An important aside is that exponentiation of compressed pairings must take
into account the fact that they are actually traces of full pairings. This means
one cannot exponentiate a pairing as if it were a simple \( \mathbb{F}_{q^{2k}} \) value; rather, one
must always handle it as a Lucas sequence element.

---

\[ \text{Rubin and Silverberg [13] use traces to compress BLS signatures, but in an entirely}
\text{different manner, and with a compression factor much closer to 1.} \]
5 A ternary exponentiation ladder

Supersingular curves in characteristic 3 are a popular choice of underlying algebraic structure for pairing-based cryptosystems, since many optimisations are possible in such a setting [1, 11, 12]. Pairing compression is possible for those systems, and we now propose a ternary ladder for Lucas sequences in characteristic 3 that keeps the exponentiation cost in \( \mathbb{F}_{q^6} \) within about 33% of the exponentiation cost in \( \mathbb{F}_{q^{k/3}} \).

Assume the sequence element index is written in signed ternary notation, \( K = (d_{t-1}, \ldots, d_0)_3 \), with \( d_{t-1} = 1 \). At step \( j \) (counting downwards from \( t-1 \) to 0), we want to compute \( V_{K_j} \) where \( K_j = \sum_{i=j}^{t-1} d_i 3^{i-j} \). Thus, by definition, \( K_j = 3K_{j+1} + d_j \).

For \( d_j = -1 \), we write down the formulas to compute \( V_{3K_{j+1}-2}, V_{3K_{j+1}-1}, \) and \( V_{3K_{j+1}} \):

\begin{align*}
V_{3K_{j+1}} &= V_{K_{j+1}}^3 \\
V_{3K_{j+1}-1} &= P V_{3K_{j+1}-2} - V_{K_{j+1}-1}^3 \\
V_{3K_{j+1}-2} &= P V_{3K_{j+1}-1} - V_{K_{j+1}}^3
\end{align*}

Similarly, for \( d_j = 1 \) we write down the formulas to compute \( V_{3K_{j+1}}, V_{3K_{j+1}+1}, \) and \( V_{3K_{j+1}+2} \):

\begin{align*}
V_{3K_{j+1}} &= V_{K_{j+1}}^3 \\
V_{3K_{j+1}+1} &= P V_{3K_{j+1}+2} - V_{K_{j+1}+1}^3 \\
V_{3K_{j+1}+2} &= P V_{3K_{j+1}+1} - V_{K_{j+1}}^3
\end{align*}

In each case, the second and third relations constitute a simple linear system. Solving them, we get these expressions for \( V_{3K_{j+1}-1}, V_{3K_{j+1}}, \) and \( V_{3K_{j+1}+1} \):

\begin{align*}
V_{3K_{j+1}-1} &= (P^2 - 1)^{-1}(PV_{K_{j+1}}^3 + V_{K_{j+1}-1}^3) \\
&= (P^2 - 1)^{-1}[(PV_{K_{j+1}}^3 + (PV_{K_{j+1}} - V_{K_{j+1}+1})^3)] \\
&= (P^2 - 1)^{-1}[(P + P_3)V_{K_{j+1}}^3 - V_{K_{j+1}+1}^3] \\
V_{3K_{j+1}} &= V_{K_{j+1}}^3 \\
V_{3K_{j+1}+1} &= (P^2 - 1)^{-1}(PV_{K_{j+1}}^3 + V_{K_{j+1}+1}^3) \\
&= (P^2 - 1)^{-1}[(PV_{K_{j+1}}^3 + (PV_{K_{j+1}} - V_{K_{j+1}+1})^3)] \\
&= (P^2 - 1)^{-1}[(P + P_3)V_{K_{j+1}}^3 - V_{K_{j+1}+1}^3]
\end{align*}

If \( (P^2 - 1)^{-1} \) and \( P + P_3 \) are precomputed, computing \( V_{3K_{j+1}} \) and one of \( V_{3K_{j+1}-1} \) or \( V_{3K_{j+1}+1} \) involves two products and two cubes, and the computation can be carried out using only \( V_{K_{j+1}} \) and one of \( V_{K_{j+1}-1} \) or \( V_{K_{j+1}+1} \). We can therefore keep track of which value between these two actually accompanies \( V_{K_{j+1}} \), and compute \( V_{K_{j}} \) and \( V_{K_{j+1}} \) at the cost of only 2 products and two
cubes per step. Besides, since we are working in characteristic 3, the cost of cubing is negligible compared to the cost of multiplying.

The binary ladder computes $V_{K_j}$ and $V_{K_{j+1}}$ at the cost of one squaring and one product, or about 1.8 product, per step. However, the step count of the ternary ladder is only about $\frac{1}{\log(3)}$ of its binary counterpart, and hence its total cost is about 70% of the binary ladder. We point out that the ternary ladder can be used for plain exponentiation in characteristic 3 as an independent technique, even in contexts where compressed pairings are not desired or not an option.

A detailed ternary ladder algorithm is described in Appendix A.

6 Compressing pairings to a third of their length

**Definition 2.** The $F_{q^2}$-trace of an element $f \in F_{q^2}$ is the value $\text{tr}(f) = f + f^q + f^{q^2} \in F_{q^2}$.

The trace is $F_{q^2}$-linear: $\text{tr}(\alpha u) = \alpha \text{tr}(u)$ for any $\alpha \in F_{q^2}$ and $u \in F_{q^2}$.

When the elliptic curve has an embedding degree $k = 6$, the Tate pairing algorithm outputs an element of $F_{q^6}$ of order $r$, where $r$ divides $q^6 - 1$, but not $q^i - 1$ for $0 < i < 6$. Now $q^6 - 1 = \Phi_1(q)\Phi_2(q)\Phi_3(q)\Phi_6(q)$. Therefore the output of the Tate pairing is an element of order $r$ which divides $\Phi_6(q) = q^2 - q + 1$.

For $q \equiv 2 \pmod{3}$, these are precisely the type of points considered in the XTR public key scheme [21] (which is based on the ideas of [8]), and all of the time/space optimizations that have been developed for this scheme [21,31] apply here as well. In particular, we note that laddering algorithms again appear to be optimal [31], and the Tate pairing output can be represented by its $F_{q^2}$-trace, and hence compressed by a factor of 3. Observe that the compressed value, being a trace, must be implicitly exponentiated using the Lenstra-Verheul algorithm [21, Algorithm 2.3,7] – the trace value per se is not even a point of order $r$.

For supersingular curves in characteristic 3 we can do better than merely take the trace – rather, it is possible to do nearly all computations without resorting to arithmetic any more complex than that on $F_{q^2}$.

6.1 Simpler arithmetic for pairing computation in characteristic 3

Let $q = 3^m$ for some $m \equiv 1,5 \pmod{6}$, let $b = \pm1$, and let $\sigma, \rho \in F_{q^6}$ be elements satisfying $\sigma^2 + 1 = 0$ and $\rho^3 - \rho - b = 0$. The modified Tate pairing on the supersingular curve $E(F_{3^m})$: $y^2 = x^3 - x + b$ is the mapping $e_r(P, Q) = f_P(\phi(Q))^{(q^6 - 1)/r}$ where $\phi : E(F_q) \to E(F_{q^6})$ is the distortion map $\phi(x, y) = (\rho - x, \sigma y)$.

Duursma and Lee showed [11, Theorem 5] that the modified Tate pairing for points $P = (\alpha, \beta)$ and $Q = (x, y)$ can be written as a product of factors of form $g = \beta y\bar{\sigma} - (\alpha + x + \rho + b)^2$. This expression can be rewritten as $g = \lambda - \mu\rho - \rho^2$, where $\mu \equiv \alpha + x + b \in F_q$ and $\lambda \equiv \beta y\bar{\sigma} - \mu^2 \in F_{q^2}$. Specifically, the Duursma-Lee algorithm to compute $f_P(\phi(Q))$ is as follows (cf. [11, Algorithm 4]):
Duursma-Lee algorithm to compute $f_P(\phi(Q))$:

\[
f \leftarrow 1 \\
\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
\quad \alpha \leftarrow \alpha^3, \ \beta \leftarrow \beta^3 \\
\quad \mu \leftarrow \alpha + x + b, \ \lambda \leftarrow \beta y \bar{\sigma} - \mu^2 \\
\quad g \leftarrow \lambda - \mu \rho - \rho^2, \ f \leftarrow f \cdot g \\
\quad x \leftarrow x^{1/3}, \ y \leftarrow y^{1/3} \\
\text{end for} \\
\text{return } f
\]

The output is an element $f \in \mathbb{F}_q$. We now show that this algorithm can be modified to compute $\text{tr}(f)$ instead, by maintaining a ladder of three values $[\text{tr}(f), \text{tr}(f \rho), \text{tr}(f \rho^2)]$. Since $f$ is initialized to 1, the initial ladder can be computed from $\rho$ alone, namely, $[\text{tr}(1), \text{tr}(\rho), \text{tr}(\rho^2)] = [0, 0, 2]$, as one readily deduces from the definition of $\rho$:

**Theorem 1.** Let $q = 3^m$ for some $m \equiv 1, 5 \pmod{6}$, and let $\rho \in \mathbb{F}_q^\times$ satisfy $\rho^3 - \rho - b = 0$. Then $\text{tr}(\rho) = 0$ and $\text{tr}(\rho^2) = 2$.

**Proof.** From $\rho^3 = \rho + b$ it follows by induction that $\rho^3^m = \rho + mb$, and hence $\rho^3^m = \rho^3^2m = \rho + 2mb$ and $\rho^3^4m = \rho + mb$, so that $\text{tr}(\rho) = (\rho + \rho^2 + \rho^4 = \rho + 2mb + \rho + mb = 0$. Moreover, $(\rho^2)^3 = (\rho^3)^2 = (\rho + nb)^2 = \rho^2 - nb \rho + n^2$, so that $\text{tr}(\rho^2) = \rho^2 + (\rho^2)^2 + (\rho^3)^2 = \rho^2 + \rho^2 - 2mb \rho + (2m)^2 + \rho^2 - mb \rho + m^2 = 2$. \hfill \Box

At each step of the loop, we compute $[\text{tr}(fg), \text{tr}(fg \rho), \text{tr}(fg \rho^2)]$ according to the following theorem:

**Theorem 2.** $\begin{bmatrix} \text{tr}(fg) \\ \text{tr}(fg \rho) \\ \text{tr}(fg \rho^2) \end{bmatrix} = A \cdot \begin{bmatrix} \text{tr}(f) \\ \text{tr}(f \rho) \\ \text{tr}(f \rho^2) \end{bmatrix}$, where $A \equiv \begin{bmatrix} \lambda & -\mu & -1 \\ -b & (\lambda - 1) & -\mu \\ -b \mu & (\mu + b) & (\lambda - 1) \end{bmatrix}$.

**Proof.** Using the $\mathbb{F}_q$-linearity of the trace and the defining property $\rho^3 = \rho + b$, we have $fg = f(\lambda \mu \rho - \rho^2) \implies \text{tr}(fg) = \lambda \text{tr}(f) - \mu \text{tr}(f \rho) - \text{tr}(f \rho^2)$. Similarly, $fg \rho = f(\lambda - \mu \rho - \rho^2) \rho = \lambda f \rho - \mu f \rho^2 - f \rho - bf \implies \text{tr}(fg \rho) = -b \text{tr}(f) + (\lambda - 1) \text{tr}(f \rho) - \mu \text{tr}(f \rho^2)$. Finally, $fg \rho^2 = -b f \rho + (\lambda - 1) f \rho^2 - bf - \mu f \rho + \mu b f \implies \text{tr}(fg \rho^2) = -\mu \text{tr}(f) - (\mu + b) \text{tr}(f \rho) + (\lambda - 1) \text{tr}(f \rho^2)$. \hfill \Box

Therefore, defining $L \equiv [L_0, L_1, L_2]^T = [\text{tr}(f), \text{tr}(f \rho), \text{tr}(f \rho^2)]^T$ and using the matrix $A$ defined above, the modified algorithm to compute pairing traces reads:

**A laddering algorithm to compute $\text{tr}(f_P(\phi(Q)))$:**

$L \leftarrow [0, 0, 2] \quad // = [\text{tr}(1), \text{tr}(\rho), \text{tr}(\rho^2)]$
for $i \leftarrow 1$ to $m$ do
\[ \begin{align*}
\alpha &\leftarrow \alpha^3, \quad \beta \leftarrow \beta^3 \\
\mu &\leftarrow \alpha + x + b, \quad \lambda \leftarrow \beta y - \mu^2 \\
L &\leftarrow A \cdot L \\
x &\leftarrow x^{1/3}, \quad y \leftarrow y^{1/3}
\end{align*} \]
end for

return $L_0$

However, to obtain a unique pairing value suitable for pairing-based protocols we need $\text{tr}(f_P(\phi(Q))^{q^6-1}/r)$ rather than $\text{tr}(f_P(\phi(Q)))$. Let $e = f_P(\phi(Q))$. The simplest (and seemingly the most efficient) way to do it is to recover $e$ from all three components of $L = [\text{tr}(e), \text{tr}(e\rho), \text{tr}(e\rho^2)]$.

We use the $F_{q^6}$-linearity of the trace and fact that $\{1, \rho, \rho^2\}$ is a basis of $F_{q^6}$ with respect to $F_{q^3}$, i.e. any element $e \in F_{q^6}$ can be written as $e = x + y\rho + z\rho^2$ where $x, y, z \in F_{q^3}$. The trick is straightforward:

1. $L_0 = \text{tr}(e) = \text{tr}(x + y\rho + z\rho^2) = x \text{tr}(1) + y \text{tr}(\rho) + z \text{tr}(\rho^2) = 2z \implies z = -L_0$.
2. $L_1 = \text{tr}(e\rho) = \text{tr}(x\rho + y\rho^2 + z(\rho + b)) = bz \text{tr}(1) + (x + z) \text{tr}(\rho) + y \text{tr}(\rho^2) = 2y \implies y = -L_1$.
3. $L_2 = \text{tr}(e\rho^2) = \text{tr}(x\rho^2 + y(\rho + b) + z(\rho^2 + b\rho)) = by \text{tr}(1) + (y + bz) \text{tr}(\rho) + (x + z) \text{tr}(\rho^2) = 2(x + z) \implies x = L_0 - L_2$.

Thus we recover $e$ from the pairing ladder essentially for free. Now one must compute $g = e^{(q^6-1)/r}$, and then take the trace of $g$. This can be efficiently done using the techniques described in [1, Appendix A2], at a cost roughly equivalent to a few extra steps of the laddering algorithm.

Each step of this laddering algorithm takes 17 $F_q$ multiplications. This compares well with the original Duursma-Lee algorithm where each step takes 20 $F_q$ multiplications, and avoids $F_{q^6}$ arithmetic in the main loop.

6.2 Implicit exponentiation in characteristic 3

It is quite commonplace that the pairing value undergoes further exponentiation as dictated by the underlying cryptographic protocol. We are thus confronted with the task of computing $\text{tr}(g^n)$ given the value of $\text{tr}(g)$. The Lenstra-Verheul algorithm [21, Algorithm 2.3.1] performs this task for characteristic $p = 2$ (mod 3). We now describe a variant tailored for characteristic 3.

Let $c \in F_{q^2}$, and let $F(c, X) = X^3 - cX^2 + cX - 1 \in F_{q^2}[X]$ with roots $h_0, h_1, h_2 \in F_{q^2}$. One can show [21, Lemma 2.2.1] that, if $g \in F_{q^2}$ is an element of order dividing $\Phi_3(q) = q^2 - q + 1$, then the roots of $F(\text{tr}(g), X)$ are the $F_{q^2}$-conjugates of $g$. Defining $c_n \equiv h_0^n + h_1^n + h_2^n$, one can further show [21, Lemmas 2.3.2 and 2.3.4] (see also [9]) that $c_{-n} = c_n^q$ and $c_{n+v} = c_n c_v - c_n^q c_{n-v} + c_{n-2v}$. The proofs of these properties are independent of the field characteristic.
From the above properties, one easily deduces the following relations that hold in characteristic 3:

\[ c_{2n} = c_n^2 + c_n^3 \]
\[ c_{3n} = c_n^3 \]
\[ c_{3n-1} = c_{2n} \cdot c_{n-1} - c_{n-1}^3 \cdot c_{n+1} + c_2 \]
\[ c_{3n-2} = c^{-q} \cdot (c_{n-1} - c_n)^3 + c^{q-1} \cdot c_{3n-1} \]
\[ c_{3n+1} = c_{2n} \cdot c_{n+1} - c_{n+1}^3 \cdot c_{n-1} + c_2^3 \]
\[ c_{3n+2} = c^{-1} \cdot (c_{n+1} - c_n)^3 + c^{q-1} \cdot c_{3n+1} \]

Computing \( c_{2n} \) takes two \( \mathbb{F}_q \) multiplications, \( c_{3n \pm 1} \) takes four \( \mathbb{F}_q \) multiplications, and \( c_{3n \pm 2} \) takes six \( \mathbb{F}_q \) multiplications.

Define \( L_n(c) \equiv (c_{3n}, c_{3n+1}, c_{3n+2}, c_{3n+3}) \in (\mathbb{F}_q^*)^3 \). Using the above formulas, one can compute any one of \( L_n(c), L_{3n+1}(c), \) or \( L_{3n+2}(c) \) from \( L_n(c) \) at the cost of 12 \( \mathbb{F}_q \) multiplications:

\[
L_{3n} = (c_{3n}, c_{3n+1}, c_{3n+2}, c_{3n+3}) = (c_{3(3n)}, c_{3(3n+1)}-2, c_{3(3n+1)}-1, c_{3(3n+1)}) \\
L_{3n+1} = (c_{3n+3}, c_{3n+4}, c_{3n+5}, c_{3n+6}) = (c_{3(3n+1)}, c_{3(3n+2)}-1, c_{3(3n+2)}-1, c_{3(3n+2)}) \\
L_{3n+2} = (c_{3n+5}, c_{3n+6}, c_{3n+7}, c_{3n+8}) = (c_{3(3n+2)}, c_{3(3n+3)}+1, c_{3(3n+3)}+1, c_{3(3n+3)})
\]

From the definition of \( c_n \), it is clear that \( c_n = \text{tr}(g^n) \) if \( c = \text{tr}(g) \). Hence, if \( L_{[n/3]}(\text{tr}(g)) = (S_0, S_1, S_2, S_3) \), then \( \text{tr}(g^n) = S_{n \mod 3} \). The total cost of this algorithm, about \( 7.61 \lg n \) \( \mathbb{F}_q \) multiplications, matches the complexity of the ternary ladder introduced in section 5 for \( \mathbb{F}_q \)-trace exponentiation. Appendix B lists this algorithm in detail. We point out that this ternary ladder can also be the basis of a characteristic 3 variant of the XTR cryptosystem.

### 6.3 Coupling pairing compression with point reduction

A nice feature of this algorithm is that it is compatible with a variant of the point reduction technique.

The conventional approach to compress a point \( R = (u, v) \) is to keep only \( u \) and a single bit of \( v \); point reduction discards \( v \) altogether. In characteristic 3, it is more advantageous to discard \( u \) instead, keeping \( v \) and a trit of \( u \) to distinguish among the solutions of the curve equation \( u^3 - u + (b - x^2) = 0 \); alternatively, one can reduce \( R \) by keeping only \( v \) and modifying the cryptographic protocols to allow for any of the three points \( R_0, R_1, \) and \( R_2 \) that share the same \( u \). Thus, we will show that the input to the laddering algorithm of section 6.1 can be only \( y \) (or \( \beta \)); the corresponding \( x \) (or \( \alpha \)) can be easily recovered except for a trit, and the actual choice of this trit does not affect the compressed pairing value.

Let \( z \in \mathbb{F}_q \) where \( q = 3^m \) for odd \( m \), and assume the order \( r \) of \( z \) divides \( \Phi_v(q) \), i.e. \( r \mid q^2 - q + 1 \). The conjugates of \( z \) are \( z, z^q, \) and \( z^{q^2} \), or equivalently \( z, z^q-1, \) and \( z^{-q} \), since \( q^2 \equiv q-1 \) (mod \( r \)) and \( q^4 \equiv -q \) (mod \( r \)). The trace of \( z \) is the sum of the conjugates, \( \text{tr}(z) = z + z^{q-1} + z^{-q} \) [21]. Consider the supersingular elliptic curve \( E : y^2 = x^3 - x + b, b \in \{-1, 1\} \), whose order is [23,
section 5.2.2] \( n = q + 1 - t = 3^m + 1 \pm 3^{(m+1)/2} \), where \( t = \pm 3^{(m+1)/2} \) is the trace of the Frobenius.

Let \( P = (x, y) \in E(\mathbb{F}_q) \), and let \( Q \in E(\mathbb{F}_{q'}) \) be a linearly independent point. The conjugates of \( e(P, Q) \) are \( e(P, Q), e(P, Q)^{q-1} = e([q-1]P, Q) \), and \( e(P, Q)^{-q} = e(-qP, Q) \). The following property holds:

**Lemma 1.** If \( P \in E[r] \), points \( P, [q-1]P, \) and \( -qP \) share precisely the same y coordinate.

**Proof.** Let \( P = (x, y) \). A simple inspection of the group law for characteristic 3 [1] reveals that \( 3P = (x^3 - b, -y^9) \), and hence \( 3^jP = (x^{3^j} - jb, (-1)^j y^{9^j}) \). Thus \( [q-1]P = q^2P = 3^{2m}P = (x^{3^{2m}} - 2mb, (-1)^{2m} y^{9^{2m}}) = (x + mb, y) \), where we used the fact that \( u^{3^m} = u \) for any \( u \in \mathbb{F}_q \). Similarly, \( -qP = q^2(q^2P) = q^2(x + mb, y) = (x - mb, y) \). \( \square \)

We see that, for \( m \not\equiv 0 \) (mod 3), the \( x \) coordinates of \( P, [q-1]P, \) and \( -qP \) are the three solutions to \( x^3 - x + (1 - y^3) = 0 \), which are exactly \{\( x, x+1, x+2 \}\). Obviously, the traces of the pairings computed from the conjugates of \( P \) are all equal, since \( \text{tr}(e(P, Q)) \) is simply the sum of the conjugates of \( e(P, Q) \). Thus, the actual solution \( x \) to the curve equation above used to compute \( \text{tr}(e(P, Q)) \) is irrelevant. Also, computing \( x \) from \( y \) is very efficient, since it amounts to solving a linear system (see appendix C).

## 7 Conclusions

We have introduced the notion of compressed pairings, and suggested how they can be realised as traces of ordinary Tate pairings. We also described how compressed pairings can be computed and implicitly exponentiated by means of laddering algorithms, with a compression ratio of 1/2 in characteristic \( p > 3 \) and 1/3 in characteristic 3; our algorithms thus reduce bandwidth requirements without impairing performance. Finally, we showed how to couple compressed pairings with the technique of point compression or point reduction. As a side result, we proposed an efficient laddering algorithm for plain exponentiation in characteristic 3, which can be used even in contexts where compressed pairings are not desired.

Our work constitutes evidence that the security of pairing-based cryptosystems is linked to the security of the Lucas/XTR schemes, and gives further motivation for the approach of Galbraith et al regarding the use of traces to prevent security losses.

We leave it as an open problem to find a method to compute pairings directly in compressed form when the compression ratio is 1/3 or better on ordinary (non-supersingular) curves in characteristic \( p > 3 \).
8 Acknowledgements

We are grateful to Steven Galbraith, Robert Granger, and Waldyr Benits Jr. for their valuable comments during the preparation of this work, and to the anonymous referees for their improvement suggestions.

References

A Computation of Lucas sequence elements

The Lucas sequence $V_n(P,1)$ for some field element $P$ is defined by the following recurrence relations:

$$V_0 = 2, \quad V_1 = P, \quad V_{n+1} = PV_n - V_{n-1}.$$ 

Let $n = (n_t \ldots n_0)_3$ be an integer in binary representation, with $n_t = 1$. The Lucas sequence element $V_n(P,1)$ can be computed as:

$$v_0 \leftarrow 2, \quad v_1 \leftarrow P$$

for $j \leftarrow t$ downto $0$ do

if $n_j = 1$ then

$v_0 \leftarrow v_0v_1 - P$, $v_1 \leftarrow v_1^2 - 2$

else

$v_1 \leftarrow v_0v_1 - P$, $v_0 \leftarrow v_0^2 - 2$

endif

end for

return $v_0$

Let $n = (n_t \ldots n_0)_3$ be the signed ternary representation of $n \geq 0$. The Lucas sequence element $V_n(P,1)$ in characteristic 3 (as needed for the implicit exponentiation of $F_{q^3}$-traces of $F_{q^6}$ values) can be computed using the following algorithm:

$$\mu \leftarrow (P^2 - 1)^{-1}, \quad T \leftarrow P + P^3$$

$v_0 \leftarrow 2, \quad v_1 \leftarrow P, \quad up \leftarrow true$

for $j \leftarrow t$ downto $0$ do

$w \leftarrow v_0^3$

if $n_j = -1$ then

$v_0 \leftarrow if up then \mu(Tw - v_1^3)$ else $Pw + v_1^3$$

$v_1 \leftarrow w$

$up \leftarrow true$

else if $n_j = 1$ then

$v_0 \leftarrow if up then \mu(Pw + v_1^3)$ else $Tw - v_1^3$$

$v_1 \leftarrow w$

$up \leftarrow false$

else /* $n_j = 0 */$

$v_1 \leftarrow if up then \mu(Pw + v_1^3)$ else $Tw - v_1^3$$

$v_0 \leftarrow w$

$up \leftarrow true$

endif

end for

return $v_0$
B Implicit exponentiation of $\mathbb{F}_{q^{\sqrt{3}}}$-traces

Let $n = (n_1 \ldots n_0)_3$ be the plain ternary representation of $n \geq 0$. The following algorithm computes the $\mathbb{F}_{q^{\sqrt{3}}}$-trace $c_n \equiv \text{tr}(g^n)$ of an element $g \in \mathbb{F}_{q^d}$ from its $\mathbb{F}_{q^{\sqrt{3}}}$-trace $c \equiv \text{tr}(g)$.

$$
c^{-1} \leftarrow c^3 \cdot (c^3 \cdot c)^{-1} \quad / / \text{N.B. } (c^3 \cdot c) \in \mathbb{F}_q
$$

$$
c^{q^{-1}} \leftarrow c^2 \cdot c^{-1}, c^{-q} \leftarrow (c^{-1})^q, c^{1-q} \leftarrow (c^{q-1})^q, c_2 \leftarrow c^2 + e^q
$$

$S_0 \leftarrow 0, S_1 \leftarrow c, S_2 \leftarrow c_2, S_3 \leftarrow c^3$

for $j \leftarrow t$ downto 0 do

if $n_j = 0$ then

$S'_0 \leftarrow S_1^3$

$S'_2 \leftarrow (S'_1 + S'_2) \cdot S_0 - S'_2 \cdot S_2 + c_2$

$S'_3 \leftarrow c^{-q} \cdot (S_0 - S_1)^3 + c_{1-q} \cdot S_2$

$S'_0 \leftarrow S'_3$

else if $n_j = 1$ then

$s_1 \leftarrow S_1$

$s_2 \leftarrow S_2$

$S'_1 \leftarrow (s_1^3 + s_2^3) \cdot s_2 - s_2^3 \cdot s_1 + c_2^3$

$S'_0 \leftarrow s_1^3$

$S'_2 \leftarrow (s_2^3 + s_2^3) \cdot s_1 - s_2^3 \cdot s_3 + c_2$

$S'_3 \leftarrow s_2^3$

else /* $n_j = 2$ */

$S'_0 \leftarrow S'_3$

$S'_1 \leftarrow (S'_2 + S'_2^3) \cdot S_3 - S'_2 \cdot S_1 + c_2^3$

$S'_2 \leftarrow c^{-1} \cdot (S_3 - S_2)^3 + c_{1-q} \cdot S'_1$

$S'_3 \leftarrow S'_3$

end if

end for

return $S_{n \bmod 3}$

C Solving the curve equation in characteristic 3

**Definition 3.** The absolute trace of a field element $a \in \mathbb{F}_{3^m}$ is the linear form:

$$
\text{tr}(a) = a + a^3 + a^9 + \cdots + a^{3^{m-1}}.
$$

The absolute trace will always be in $\mathbb{F}_3$ as one can easily check by noticing from the above definition that $\text{tr}(a^3) = \text{tr}(a)$, for all $a \in \mathbb{F}_{3^m}$. Being surjective and linear over $\mathbb{F}_3$, it can always be represented as a (usually sparse) dual vector $T \in \mathbb{F}_{3^m}$ in a given basis, so that one can compute $\text{tr}(u) = T \cdot u$ in no more than $O(m)$ time. In a normal basis $\{\theta^a\}$ with $\text{tr}(\theta) = 1$, computing $\text{tr}(u)$ amounts to summing up all coefficients of $u$.

The coordinates of a curve point $P = (x, y)$ are constrained by the curve equation to satisfy $y^2 = x^3 + ax + b$. Thus one can represent a point as either $(x, \beta)$
where \( \beta \in \mathbb{F}_2 \) indicates which of the two roots correspond to \( y = \pm \sqrt[3]{x^3 + ax + b} \), or else by \( (\tau, y) \) where \( \tau \in \mathbb{F}_3 \) indicates which of the three solutions one has to take of the equation \( x^3 + ax + (b - y^2) = 0 \). In characteristic 3, cubing is a linear operation, which makes the second possibility more advantageous.

Consider the special equation \( x^3 - x - u = 0 \) for a given \( u \in \mathbb{F}_3^m \), which is relevant for supersingular curves in characteristic 3. This equation has a solution if, and only if, \( \text{tr}(u) = 0 \) [22, theorem 2.25]. This is the case for \( 1/3 \) of the elements in \( \mathbb{F}_3^m \), since the trace function is linear and surjective. The complexity of solving the cubic equation is only \( O(m^2) \), as we show now.

Let \( C : \mathbb{F}_3^m \rightarrow \mathbb{F}_3^m \) be defined by \( C(x) = x^3 - x \). The kernel of \( C \) is \( \mathbb{F}_3 \) [22, chapter 2 section 1], hence the rank of \( C \) is \( m - 1 \) [16, section 3.1, theorem 2].

**Theorem 3.** The equation \( x^3 - x - u = 0 \) over \( \mathbb{F}_3^m \) can be solved in \( O(m^2) \) steps.

**Proof.** If \( \mathbb{F}_3^m \) is represented in standard polynomial basis, the cubic equation reduces to a system of linear equations with coefficients in \( \mathbb{F}_3 \), and can be solved in no more than \( O(m^2) \) steps. This is achieved by first checking whether the system has solutions, i.e. whether \( \text{tr}(u) = 0 \). If so, since the rank of \( C \) is \( m - 1 \) one obtains an invertible \( (m - 1) \times (m - 1) \) matrix \( A \) by leaving out the one row and corresponding one column of the matrix representation of \( C \) on the given basis. A solution of the cubic equation is then given by an arbitrary element \( x_0 \in \mathbb{F}_3 \) and by the solution of system \( Ax = \bar{u} \), which is obtained as \( x = A^{-1} \bar{u} \) in \( O(m^2) \) time.

Using a normal basis to represent field elements, it is not difficult to see that the cubic equation can be efficiently solved in \( O(m) \) time by the following algorithm (the proof is straightforward and left as an exercise):

**Cubic equation solving in normal basis:**

\[
x_0 \leftarrow \text{root selector (an arbitrary element from} \mathbb{F}_3 \text{)}
\]

\[\text{for } i \leftarrow 1 \text{ to } m - 1 \text{ do } \{
\]
\[x_i \leftarrow x_{i-1} - u_i
\]
\[\}
\]

\( x \) is a solution if, and only if, \( x_{m-1} = x_0 + u_0 \).

\[\square\]