Pairings for Cryptographers *

Steven D. Galbraith[1], Kenneth G. Paterson[1], and Nigel P. Smart[2]

[1] Information Security Group,
Royal Holloway, University of London,
Egham, Surrey, TW20 0EX,
United Kingdom.
{steven.galbraith,kenny.paterson}@rhul.ac.uk
[2] Department of Computer Science,
University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB,
United Kingdom. nigel@cs.bris.ac.uk

Abstract. Many research papers in pairing based cryptography treat pairings as a "black box". These papers build cryptographic schemes making use of various properties of pairings. If this approach is taken, then it is easy for authors to make invalid assumptions concerning the properties of pairings. The cryptographic schemes developed may not be realizable in practice, or may not be as efficient as the authors assume. The aim of this paper is to outline, in as simple a fashion as possible, the basic choices that are available when using pairings in cryptography. For each choice, the main properties and efficiency issues are summarized. The paper is intended to be of use to non-specialists who are interested in using pairings to design cryptographic schemes.

1 Introduction

The use of pairings in cryptography has developed at an extraordinary pace since the publication of the paper of Joux, [10]. For example, there have been papers on identity based encryption [4, 14–16, 2, 8], short signatures [5], group signatures [7, 3], and many more. Many research papers in the field treat pairings as a "black box" and then proceed to build various cryptographic schemes making use of assumed properties of the pairings. This is not necessarily a bad approach, since the details of pairings, particularly their selection and implementation, can be quite complex. As an approach, it allows one to ignore mathematical and algorithmic subtleties and focus on purely cryptographic aspects of the research.

However, if this approach is taken, then it is easy for authors to make assumptions concerning the properties of pairings which are not necessarily correct,

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

and hence develop cryptographic schemes which cannot realized in practice, or which cannot be implemented as efficiently as the authors assume. Some common assumptions of this type are as follows:

- One can efficiently hash onto groups associated with the pairing.
- Operations in groups associated with the pairing can be efficiently implemented.
- Elements of one or more groups associated with the pairing have a "short" representation.
- One can construct suitable system parameters for pairing-based cryptosystems in polynomial time for any security level.
- The pairing can be computed efficiently.
- There are efficiently computable homomorphisms between various groups associated with the pairing.

In general, it is easy to set up systems satisfying some of these assumptions. But, as we shall see, it is not true that pairings can be constructed so that all of these assumptions hold simultaneously.

Certainly, many researchers in pairings and pairing-based cryptography are aware of these difficulties, but it appears that many more are not.[1] The aim, then, of the present article is to give a summary of the different ways that pairings can be implemented, along with the properties of each choice (with respect to the above list of assumptions) and the pros and cons of each choice. To make the article accessible to as large an audience as possible, we attempt to keep the presentation largely free of technical jargon. However, some technicalities will be necessary in order to explain the choices that are available to cryptographers, and we do provide brief explanatory notes justifying our ratings. For a full presentation of the details of pairing-based cryptography we recommend consulting [1, Chapters IX and X].

Two final *caveats* need to be stated. Firstly, our recommendations are a simplification of a complicated subject that is still in a state of flux. It is very likely that the future will bring new methods to deal with the issues raised here. Furthermore, reaching a consensus on the relative merits of one implementation of pairings over another is very difficult. Hence, our findings should be considered more as a summary of the limitations of present methods, rather than as a recipe for the 'best' way to implement pairings in practice.

Secondly, this article attempts neither to introduce new research, nor to act as a survey of the area. Our goal is to present in a simple manner the distinct choices which need to be made when designing systems based on pairings. We have been led to write this article by the large number of questions we have had from protocol designers in the last few years who have had problems in this area. This problem arises due to the plethora of papers in the area, which use a variety of slightly different assumptions about the underlying pairing groups. Some of these different assumptions are made for good technical reasons, and others are just for presentational purposes. In this paper we have attempted to present, in

[1] It is not our intention to "name and shame" such researchers, though it is tempting!

the simplest manner possible, the basic choices which a cryptographic designer needs to make.

The remainder of this article is organized as follows. In the next section, we provide further introductory background, then give a classification of the main ways of realizing pairings for cryptographic applications along with the main properties of each choice. Section 3 develops the analysis to include more refined implementation issues (bandwidth usage, efficiency of implementation and their relationships with security). We use a simple but subjective rating system to score each option at a variety of different security levels. The appendix contains technical notes justifying the ratings in Section 3.

2 Background

There are two forms of pairings used in the cryptography literature. The first are of the form

$$e : G_1 \times G_1 \longrightarrow G_T$$

where G_1 and G_T are groups of prime order l. (See note 2 of the Appendix for the case of composite group order.) The second form is

$$e : G_1 \times G_2 \longrightarrow G_T$$

where G_1, G_2 and G_T are groups of prime order l. (See note 1 of the Appendix for another variant.) We will always use the second form, and we consider the first form to be just the special case $G_2 = G_1$. Of course this important special case may yield advantages in practice (but as we shall see, it is actually one of the least flexible options).

One of the main goals of the paper is to explore various issues which arise depending on the choices of groups and pairing. We comment that the groups G_1, G_2, G_T and the pairing $e(\cdot, \cdot)$ often form part of the system parameters of a cryptosystem and may be used by a large number of users. For example, in many identity-based encryption schemes, the trusted authority sets up the system parameters which includes descriptions of groups and a pairing, and all users' public keys are defined with respect to these parameters.

It turns out to be appropriate to separate different possible pairing instantiations into three basic types:

Type 1: $G_1 = G_2$;
Type 2: $G_1 \neq G_2$ but there is an efficiently computable homomorphism $\phi : G_2 \rightarrow G_1$;
Type 3: $G_1 \neq G_2$ and there are no efficiently computable homomorphisms between G_1 and G_2.

We should clarify that in all cases, there exist homomorphisms between G_1 and G_2 (this is trivially true since they are cyclic groups of the same order) but that computing these homomorphisms is presumably as hard as computing discrete logarithms in the groups. Type 2 is when there is an efficiently computable homomorphism from G_2 to G_1 and there is not an efficiently computable homomorphism from G_1 to G_2. The situation where $G_1 \neq G_2$ but there are efficiently computable homomorphisms in both directions can be re-interpreted as Type 1, so we do not consider it separately.

This distinction into types is relevant for the design of cryptographic schemes. In particular, the existence of maps between G_2 and G_1 is sometimes required to get a security proof to work (see for example [6] and [17] for a general discussion on this point). There exist many primitives in pairing-based cryptography whose security proof does not apply if the cryptosystem is implemented using pairings of the third type.

We now focus on several of the frequently made assumptions about pairings when they are treated as black boxes.

- One can hash to G_2.
- There is a (relatively) short representation for elements of G_1.
- There is an efficiently computable homomorphism from G_2 to G_1.
- One can generate system parameters (including groups and a pairing) achieving at least κ bits of security, where κ is a security parameter, in time polynomial in κ.

Many authors assume that some or all of these properties can easily be achieved. However, it is not true that all these properties can simultaneously be achieved.

We briefly summarise what is possible in Table 1, but first we should mention some technical properties of pairing implementations (it is not necessary for the reader to understand what these terms mean). The Type 1 case $G_1 = G_2$ is implemented using supersingular curves. The supersingular curves can be separated into two sub-classes: those over fields of characteristic 2 or 3 (with embedding degree 4 or 6 respectively), and those over fields of large prime characteristic (with embedding degree 2). The curves of Type 2 are ordinary and the homomorphism from G_2 to G_1 is the trace map. The curves of Type 3 are ordinary, and G_2 is typically taken to be the kernel of the trace map.

A ✓ in Table 1 indicates that there exist implementations of pairings for which the property holds. A $\times$ indicates that there is no known method to implement pairings of the stated type which achieve the required property (though a $\times$ in the Type 1 (small characteristic) row denotes the slightly weaker claim that there is no good solution for almost all security levels).

A more detailed analysis of these issues will be given below. The most important message is that it is currently impossible to simultaneously have all the frequently used features of pairings. We hope that Table 1 will be useful for cryptographic designers working with pairings. Once the designer has identified the required properties for their system, they can consult the table to determine the possible (if any) instantiations of their system.

2.1 Issues Related to Provable Security

Provable security typically studies the asymptotic security of cryptosystems as a security parameter κ goes to infinity. Hence, to apply the provable security

Table 1. Properties of the types of pairing groups.

Type	Hash to G_2	Short G_1	Homomorphism	Poly time generation
1 (small char)	✓	×	✓	×
1 (large char)	✓	×	✓	✓
2	×	✓	✓	✓
3	✓	✓	×	✓

paradigm it is necessary that there is an algorithm to generate system parameters which takes as input a security parameter κ and runs in time polynomial in κ.

As we have noted above, the existence of a polynomial time algorithm for the generation of system parameters is not automatic. Hence, asymptotic security results for cryptographic schemes only give heuristic evidence for security when the scheme is implemented using general pairings.

None of this is necessarily a problem in practice, since it is usually possible to generate parameters offering a required level of security efficiently and with a reasonable degree of flexibility. We will study in the next part the question of whether there are sufficiently many curves available and whether key generation is a problem in practice.

3 Efficiency and Bandwidth Considerations

Once one has decided, with the help of Table 1, whether a proposed scheme can be implemented, it is natural to ask about the speed and storage requirements of the system. We discuss these issues in this section. It turns out that these properties can change as the security level increases. The results depend on specific implementation details of the relevant group operations and pairing calculation, and so this section requires a little bit more technical discussion than the previous sections.

It is necessary to discuss, for each of the three types defined above, how to ensure an appropriate security level is attained.

First we note that all practical pairings are based on the Weil pairing or Tate pairing on elliptic (or hyperelliptic) curves over finite fields. In this paper we restrict to elliptic curves. The groups G_1 and G_2 are groups of points on the curve and the group G_T is a subgroup of the multiplicative group of a related finite field. We write l for the (common) order of these three groups. If q denotes the size of the field over which our elliptic curve E is defined, then G_1 is a subgroup of $E(\mathbb{F}_q)$, G_2 is usually a subgroup of $E(\mathbb{F}_{q^k})$, and G_T is a subgroup of $\mathbb{F}_{q^k}^*$. Here k is a parameter usually called the *embedding degree* in pairing-based cryptography. There are then three main parameters that one needs to keep in mind: the base field size q, the embedding degree k and the group size l.

Secondly, we note that in order to achieve a particular level of security, it is necessary that the discrete logarithm problems (DLPs) in G_1, G_2 and G_T be adequately hard. Thus we need to consider (as a first step) what minimum sizes we need our base field $\mathbb{F}_q$ and our extension field $\mathbb{F}_{q^k}$ to be in order to make the relevant DLPs sufficiently hard. Even this is a complicated question, particularly with regard to selecting $\mathbb{F}_{q^k}$, as there is a variety of algorithms for solving the DLP, and these algorithms have complicated asymptotic running times.

We can simplify the discussion by referring to Table 2, which shows roughly equivalent parameter sizes at a variety of security levels from three different sources, NIST [18], Lenstra [11], and ECRYPT [19]. The first column in this table shows the security level κ. Roughly speaking, 2^κ is the number of basic operations (block cipher encryptions, say) needed to break a block cipher with a κ-bit key. The second column represents the size of an elliptic curve group needed to provide κ bits of security (again, meaning that 2^κ basic operations are needed to solve the DLP in the group). Note the simple relationship between κ and the group size $2^{2\kappa}$. The third column shows the size of RSA keys needed to provide κ bits of security. This can be roughly equated to the size of field needed to attain a given level of security for the DLP in $\mathbb{F}_{q^k}$.

Table 2. Recommend Key Sizes

Author	κ	ECC-style	RSA-style
NIST[18]	80	160	1024
	128	256	3072
	256	512	15360
Lenstra[11]	80	160	1329
	128	256	4440
	256	512	26268
ECRYPT[19]	80	160	1248
	128	256	3248
	256	512	15424

An important point, which some researchers in pairing based cryptography may not be aware of, is that the efficiency of systems using pairings scales more-or-less like RSA rather than like ECC. Hence, cryptosystems using pairings often do not have the same advantages over RSA as one might expect from a cryptosystem using elliptic curves.

For example, if we decide to use NIST's figures, then to achieve 256 bits of security, we will need to select a subgroup of $E(\mathbb{F}_q)$ having size at least 512 bits. This means that q will be at least 512 bits in size. We also need to ensure that $\mathbb{F}_{q^k}$ is at least 15360 bits in size. It follows that any protocol which computes or transmits pairing values will have performance constrained by the operations in the 15360-bit finite field.

Another subtle point is that it is often difficult to generate appropriate system parameters for pairing based schemes. Continuing the above example, one might assume that l and q can be chosen to be of 512 bits in size, and so k would be chosen to be $15360/512 = 30$. However, there is currently no way known to generate elliptic curves with embedding degree 30 and with $l \approx q \approx 2^{512}$. A full discussion of these issues is beyond the scope of the present paper.

In Table 3 we answer the following questions, at the 80-bit and 256-bit security levels. (Notes to the table can be found in the appendix.)

H1: Can one hash to G_1 efficiently?
H2: Can one hash to G_2 efficiently?
S1: Is there a short representation for elements of G_1? (Meaning, in a system with security level κ, can elements of G_1 be represented with roughly the minimum number, say $\leq 2\kappa + 10$, of bits?)
S2: What is the ratio of the size of the representation of elements of G_2 to the size of the representation of elements of G_1?
E1: Are group operations in G_1 efficient? (Meaning, in a system with security level κ, are operations in G_1 efficient when compared with usual elliptic curve cryptography in a group with security level κ?)
E2: What is the ratio of the complexity of group operations in G_2 to the complexity of group operations in G_1?
E3: What is the ratio of the complexity of group operations in G_T to the complexity of those in G_1?
P: Is the pairing efficient? (Meaning, how does the speed of pairing computation compare with alternative groups of the same security level?)
F: Is there wide flexibility in choosing system parameters? (Meaning, is it necessary for all users to share one curve, or is there plenty of freedom for users to generate their own curves of any desired security level κ?)

Questions H1, H2, S1, E1, P and F will be answered by a rating of 0 to 3 stars. Zero stars means that the operation is impossible, 1 star means the operation is possible but that there is some significant practical problem with it, 2 stars means there is a satisfactory solution, 3 stars means the question is answered as well as could be expected.

For Type 3 curves it is necessary to define the quantity e. Let D be the CM discriminant used to construct the elliptic curve. If $D = -4$ then set $e = k/\gcd(k, 4)$, if $D = -3$ then set $e = k/\gcd(k, 6)$, while if $D < -4$ then set $e = k/\gcd(k, 2) = k/2$.

Note that since we have not given absolute times/sizes in the starred columns it is difficult to compare the various types of pairing groups. This is a deliberate choice on our part, since the type of pairing group one chooses is dictated more by the scheme and hence by Table 1. Thus Table 3 is primarily meant to indicate what happens as the security level increases for a particular type of curve.

Some particular phenomena are clearly indicated in the tables. For example, Type 3 is the only choice which offers good performance and flexibility for high security parameters, and yet this choice does not permit a homomorphism from G_2 to G_1. Hence, it would be desirable if protocol designers could prove the security of their schemes without requiring such a homomorphism.

Table 3. Comparison of efficiency and bandwidth properties.

Type	κ	H1(3)	H2(3)	S1	S2(4)	E1	E2(5)	E3(6)	P	F
Type 1 (char 2)	80	$\star\star\star$	$\star\star\star$	$\star\star$	1	$\star\star$	1	8/7	$\star\star\star$	$\star$
	256	$\star$	$\star$	$\star$	1	$\star$	1	8/7	$\star$	$\star$
Type 1 (char 3)	80	$\star\star\star$	$\star\star\star$	$\star\star\star$	1	$\star\star\star$	1	3	$\star\star\star$	$\star$
	256	$\star$	$\star$	$\star$	1	$\star$	1	3	$\star$	$\star$
Type 1 (char p)	80	$\star\star$	$\star\star$	$\star$	1	$\star$	1	1/4	$\star\star\star$	$\star\star\star$
	256	$\star$	$\star$	$\star$	1	$\star$	1	1/4	$\star$	$\star\star\star$
Type 2	80	$\star\star\star$		$\star\star\star$	k	$\star\star\star$	k^2	$k^2/16$	$\star$(10)	$\star\star\star$
	256	$\star\star/\star\star\star$(7)		$\star/\star\star\star$(8)	k	$\star\star/\star\star\star$(9)	k^2	$k^2/16$	$\star$(10)	$\star\star\star$
Type 3	80	$\star\star\star$	$\star$	$\star\star\star$	e	$\star\star\star$	e^2	$k^2/16$	$\star\star\star$	$\star\star\star$
	256	$\star\star/\star\star\star$(7)	$\star$	$\star/\star\star\star$(8)	e	$\star\star/\star\star\star$(9)	e^2	$k^2/16$	$\star\star\star$	$\star\star\star$

4 Conclusions

Tables 1 and 3 give a brief and non-technical guide to the black boxes which are available for protocol designers who want to use pairings. We hope that they will be useful to the designers of cryptographic schemes.

Acknowledgements

We would like to thank Dan Boneh, Eike Kiltz, Mike Scott and Hovav Shacham for useful comments and suggestions.

References

1. I. Blake, G. Seroussi and N. P. Smart (eds.). *Advances in elliptic curve cryptography.* Cambridge University Press, 2005.
2. D. Boneh and X. Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *Advances in Cryptology - Eurocrypt 2004*, Springer-Verlag LNCS 3027, 223–238, 2004.
3. D. Boneh, X. Boyen and H. Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO 2004*, Springer-Verlag LNCS 3152, 41–55, 2004.
4. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology – CRYPTO 2001*, Springer-Verlag LNCS 2139, 213–229, 2001.
5. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology – ASIACRYPT 2001*, Springer-Verlag LNCS 2248, 514–532, 2001.
6. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. of Cryptology*, **17**, 297–319, 2004.
7. D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In *ACM CCS 2004*, 168-177, ACM Press, 2004.

8. L. Chen and Z. Cheng. Security proof of Sakai-Kasahara's identity-based encryption scheme. In *Proceedings of Cryptography and Coding 2005*, Springer-Verlag LNCS 3796, 442–459, 2005.
9. G. Frey, H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. Comp.*, **62**, 865–874, 1994.
10. A. Joux. A one round protocol for tripartite Diffie–Hellman. In *Algorithmic Number Theory Symposium – ANTS IV*, Springer-Verlag LNCS 1838, 385–394, 2000.
11. A.K. Lenstra. Key lengths. In *Handbook of Information Security*, Vol 2, 617–635, Wiley, 2005.
12. F. Luca and I. Shparlinski. Elliptic curves with low embedding degree. To appear *J. of Cryptology.*
13. A. J. Menezes, T. Okamoto and S. A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Inf. Theory*, **39**, 1639–1646, 1993.
14. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems based on pairing. In *The 2000 Symposium on Cryptography and Information Security*, Okinawa, Japan, January 2000.
15. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems based on pairing over elliptic curve (in Japanese). In *The 2001 Symposium on Cryptography and Information Security*, Oiso, Japan, January 2001.
16. R. Sakai and M. Kasahara. ID based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054. 2003.
17. N.P. Smart and F. Vercauteren. On computable isomorphisms in efficient pairing based systems. Cryptology ePrint Archive, Report 2005/116. 2005.
18. NIST Recommendation for Key Management Part 1: General, NIST Special Publication 800-57. August, 2005. Available from `http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf`.
19. ECRYPT Yearly Report on Algorithms and Keysizes (2004), March 2005, Available from `http://www.ecrypt.eu.org/documents/D.SPA.10-1.1.pdf`

A Notes

Here we give technical details to justify the claims in the sections above.

For Type 1 we only consider supersingular elliptic curves with embedding degree $k = 6$ (in characteristic 3), $k = 4$ (in characteristic 2), or $k = 2$ (for large prime characteristic). Hence we do not consider the cases $k = 1$ or $k = 3$ with large prime characteristic (these cases are not very thoroughly studied, but it is clear that from a high-level view their behaviour is broadly comparable to the case $k = 2$). Similarly, we do not consider supersingular hyperelliptic curves, as from a high-level point of view their performance characteristics are similar to the case of supersingular elliptic curves.

For Types 2 and 3 we consider ordinary curves which will be generated using the CM method. There are many papers in the literature on this topic, and a wide choice of curves is available. The main focus of research has been trying to get $l \approx q$ so that one can represent elements of G_1 in an optimal way. This has not been achieved for all values of k and more research on this topic is welcome. But we feel that a sufficiently flexible array of curves is available nowadays so that implementors could get an acceptable size of elements of G_1 for any large security level. A point however which is overlooked is that for large security levels and certain choices of (k, q) it is not always the case that $l \approx q$ is the optimal in terms of efficiency.

Among the various methods for generating ordinary curves, some simply require evaluating one or more polynomials at integer values until primes are found, while others require the solution of Pell equations or finding large prime factors of $l^k - 1$. Any method for generating system parameters which involves solving Pell equations has dubious theoretical merits, since only finitely many solutions will be expected [12]. Similarly, any method that requires factoring will not be polynomial time. Hence, to ensure flexibility in the choice of parameters we assume that curves are generated using methods which only require that certain polynomials represent primes.

1. One can choose G_2 to be the full l-torsion subgroup of the curve. In other words, we have a group of exponent l rather than order l. In such a setting one obtains a tick in every column of Table 1, however this is at the expense of having a pairing between groups which has a probability of $1/l$ of being trivial on random non-trivial input elements. In addition such pairing systems consume more bits to represent the elements in G_2 compared to our other systems.
2. In a number of recent papers, pairings have been used on groups of composite order where the factors of group order are kept secret. We do not focus on these groups in this paper. However, we note that currently the only known way to generate such groups is in the Type 1 setting. In addition, such groups necessarily comsume greater bandwidth and computational resources than the "traditional" pairing systems considered in this paper.
3. Hashing into G1 and G2 usually involves multiplication by the cofactor, though in many cases this will be chosen to be small. In some schemes the need for this multiplication can be effectively removed by taking care of it through other operations at a later stage in the operation of a scheme, for example through the final powering in the Tate pairing algorithm. In these columns we assume that the cofactor multiplication is carried out.
4. We assume that $G_1 \subset E(\mathbb{F}_q)$ and $G_2 \subset E(\mathbb{F}_{q^k})$ and so the standard representation of elements of G_2 will be k times longer than the standard representation of elements of G_1. This memory requirement can be reduced in the case where G_2 is the trace zero subgroup by using twists. This is why the smaller ratio e appears for Type 3 groups. We assume for Type 3 groups that the embedding degree k is always even, so e is at most $k/2$.
5. We assume projective coordinates are used in the group G_2, rather than affine coordinates. This might not be the most efficient in any given implementation, but does give a rough order of magnitude difference.

   As explained in point 4 above, the ratio of the size of elements of G_2 to G_1 for Type 2 and 3 curves is k or e. Since multiplication is quadratic we make the naive calculation that the cost of operations in G_2 is either k^2 or e^2 the cost of operations in G_1.

If one is using pairing friendly fields, which are fields of degree $k = 2^i 3^j$, then the value of k^2 (respectively e^2) can be replaced by $3^i 5^j$ (respectively $3^{i-2}5^j$ or $3^{i-1}5^{j-1}$ or $3^{i-1}5^j$).

6. We assume a standard naive implementation as we only aim to give a rough estimate. Thus multiplication in $\mathbb{F}_{q^k}$ costs k^2-multiplications in $\mathbb{F}_q$, whereas projective coordinate addition in $G_1 \subset E(\mathbb{F}_q)$ cost roughly
   - For Type 1 curves in characteristic 2 at most 14-$\mathbb{F}_q$ operations.
   - For Type 1 curves in characteristic 3 at most 12-$\mathbb{F}_q$ operations.
   - For Type 1 curves in characteristic p at most 16-$\mathbb{F}_q$ operations.
   - For Type 2 and Type 3 curves at most 16-$\mathbb{F}_q$ operations.

   Hence the ratio of the cost of an operation in $\mathbb{F}_{q^k}$ to the cost of an operation in G_1 is $k^2/16$, for Type 2 and Type 3 curves. The values for Type 1 curves are obtained as $4^2/14$, $6^2/12$ and $2^2/16$.

   A similar comment related to pairing friendly fields as in point 5 can also be applied here.

   A common operation in the groups is exponentiation/point multiplication. Comparing the relative costs of these methods is less easy, since there are a number of special tricks available, the exact trick which is used depends on the relative cost of operations in the group, the amount of available memory, and the size of the exponent/multiplier being used.
7. When hashing into G_1 this will be efficient when k is chosen so that $q \approx l$, but when q is much larger than l then this will become progressively more expensive. Hence, this depends on k and whether curves can be generated with the correct parameter sizes.
8. This too depends on whether $q \approx l$, and hence depends on the choice of k and whether curves can be generated with the correct parameter sizes.
9. Again this depends on whether $q \approx l$, and hence depends on the choice of k and whether curves can be generated with the correct parameter sizes.
10. One can reduce a Type 2 pairing computation to that of a Type 3 pairing at the cost of an extra multiplication in G_1. One uses the following property of the pairing, if $P \in G_1$ and $Q \in G_2$ in the Type 2 situation then

$$e(P, Q) = e(P, Q - \frac{1}{k}\mathrm{Tr}(Q))$$

   where Tr is the trace function from $E(\mathbb{F}_{p^k})$ down to $E(\mathbb{F}_p)$, i.e. the function ϕ. The pairing on the right is such that its arguments are values of the pairing in the Type 3 situation.