Algorithms for Outsourcing Pairing Computation

Aurore Guillevic[2,3](✉) and Damien Vergnaud[1]

[1] Département d'Informatique, École normale supérieure, Paris, France
[2] Inria, Paris, France
aurore.guillevic@ens.fr
[3] École Polytechnique/LIX, Palaiseau, France

Abstract. We address the question of how a computationally limited device may outsource pairing computation in cryptography to another, potentially malicious, but much more computationally powerful device. We introduce two new efficient protocols for securely outsourcing pairing computations to an untrusted helper. The first generic scheme is proven computationally secure (and can be proven statistically secure at the expense of worse performance). It allows various communication-efficiency trade-offs. The second specific scheme – for optimal Ate pairing on a Barreto-Naehrig curve – is unconditionally secure, and do not rely on any hardness assumptions. Both protocols are more efficient than the actual computation of the pairing by the restricted device and in particular they are more efficient than all previous proposals.

1 Introduction

Pairings (or bilinear maps) were introduced in cryptography in 2000 by Joux [14] and Boneh-Franklin [4]. A pairing is a bilinear, non-degenerate and computable map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. In practice, the first two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are prime-order r subgroups of the group of points $E(\mathbb{F}_q)$ of an elliptic curve E defined over a finite field $\mathbb{F}_q$. The so-called *target* group $\mathbb{G}_T$ is the order r subgroup of a finite field extension $\mathbb{F}_{q^k}$. Bilinear pairings proved to be an amazingly flexible and useful tool for the construction of cryptosystems with unique features (*e.g.* efficient identity based cryptography or short signatures). However, the pairing computation is more resource consuming compared to a scalar multiplication on the elliptic curve $E(\mathbb{F}_q)$.

In the last decade, several papers [7,9,12] studied the question of how a computationally limited device may outsource pairing computation to another, potentially malicious, but much more computationally powerful device. In this setting, one wants to efficiently delegate the computation of a pairing $e(\mathcal{SK}_1, \mathcal{SK}_2)$ of two secret keys, or a pairing $e(\mathcal{SK}, \mathcal{PP})$ of a secret key and some public parameter. Obviously, one needs to ensure that this malicious device does not learn anything about what it is actually computing (*secrecy*) and sometimes one also needs to, when possible, detect any failures (*verifiability*, also called correctness).

M. Joye and A. Moradi (Eds.): CARDIS 2014, LNCS 8968, pp. 193–211, 2015.
DOI: 10.1007/978-3-319-16763-3_12

As mentioned in [7,9], a delegation protocol that does not ensure verifiability may cause severe security problems (in particular if the pairing computation occurs in the verification algorithm of some authentication protocol). Unfortunately, the different proposals for verifiable pairing delegation are very inefficient and it is actually better in practice to directly embed the pairing computation inside the restricted device than using these solutions. The main interest is then to save of area that is required to implement a pairing in the restricted device such as smart card.

However, if verifiability is mandatory in authentication protocols, this is not necessarily the case in scenarios where the delegated pairing value is used in an encryption scheme as a session key. In this case, one can indeed use additional cryptographic techniques to ensure that the values returned by the powerful device are correct (e.g. by adding a MAC or other redundancy to the ciphertext). One can even consider settings where the powerful device actually learns the pairing value. For instance, in a pay-TV scenario, the set-up box (provided by the pay-TV company) needs to know the (one-time) session key $\mathcal{K}$ used to decipher the content (e.g. movie, football match) but it does not know the secret key $\mathcal{SK}$ securely stored in the smartcard. If the smartcard delegates the pairing computation to the set-up box there is no harm to let it know the session key $\mathcal{K}$ since it will learn it anyway.

In 2005, Girault and Lefranc [12] introduced the first secure pairing delegation protocol through the *Server-Aided Verification* notion which consists in speeding up the verification step of an authentication/signature scheme. Their pairing delegation protocol only achieves secrecy with unconditional security (and the verifiability is achieved *via* a different mean). Chevallier-Mames, Coron, McCullagh, Naccache and Scott introduced in 2005 the security notions for pairing delegation [8,9] and they provided a verifiable delegation protocol for pairing computation. Their protocols are much more resource consuming for the restricted device than directly computing the pairing.

Recently, Canard, Devigne and Sanders proposed a more efficient protocol for verifiable pairing delegation. The authors showed that their proposal is more efficient than the computation of the pairing for optimal ate pairing on a so-called KSS-18 curve [15]. Unfortunately, we will show in this paper that this is not the case for state-of-the-art optimal Ate pairing on a Barreto-Naehrig curve [3].

Contributions of the paper. We propose two new efficient protocols for secret pairing delegation. Our protocols are not verifiable but as explained above, this is not really an issue for encryption primitives where verifiability can be achieved by other means. In particular, our typical usecases are Pay-TV where a smartcard delegates a pairing computation to the set-up box and encrypted GSM communication where the sim-card delegates a pairing computation to the smartphone processor (e.g. an ARM or Intel processor with high competitive performances). In these scenarios, one can even assume that the set-up box or the smartphone actually learns the pairing value (but of course not the secret information stored by the smartcard or the sim-card). Both methods enable to delegate the computation of a pairing $e(\mathcal{SK}, \mathcal{PP})$ of a secret key $\mathcal{SK}$ and some public parameter $\mathcal{PP}$. They achieve better efficiency than actual computation of the pairing by the restricted device and in particular they are more efficient than all previous proposals.

We first present a (generalized) knapsack-based approach which uses different endomorphisms on the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ to speed-up the method. Instead of masking the secret point $\mathcal{SK}$ by a scalar multiplication with a random secret exponent, it is masked by adding to it a sum of (small) multiple of random points that are also sent to the powerful device. It computes several pairings of these points with the public parameter $\mathcal{PP}$ and the restricted device combines them to get the actual value. The method is generic and can be applied to any pairing instantiation. The method increases the communication complexity between the two devices but one can propose different communication-efficiency trade-off.

In our second approach, we present a way to delegate only the non-critical steps in the pairing algorithm, looking carefully at each instruction in Miller algorithm. The powerful device does not learn any information on the secret point $\mathcal{SK}$ except the actual value of the pairing $e(\mathcal{SK}, \mathcal{PP})$ (which is perfectly suitable in our usecases). The technique can be applied to any instantiation of pairings but we concentrate on the state-of-the-art optimal Ate pairing on a Barreto-Naehrig curve [3]. We obtain a 65 % improvement (for a 128-bit security level) for the restricted device compared to the computation of the pairing.

2 Preliminaries

Timing Estimates Using the Relic Library. To illustrate the algorithms presented in this paper, we estimate the various costs of scalar multiplication, exponentiations and pairings. We choose as a practical example a Barreto–Naehrig (BN) curve [3] at the 128-bit security level with the implementation provided in Relic library of Aranha [1].

This library is at the state of the art for pairing computation [2] and is freely available for research purpose. We assume that scalar multiplications $[a]P$ and exponentiations z^a are performed with a binary signed representation of a. So it requires roughly $\log a$ doublings (resp. squarings) and $\log a/3$ additions (resp. multiplications). A doubling on a BN curve (with $a_4 = 0$) costs $2M_p + 5S_p$ (multiplications and squarings in a finite field $\mathbb{F}_p$) and a mixed addition costs $7M_p + 4S_p$ [1]. We assume that $M_p = S_p$ first because it is the case for Relic library and secondly to simplify (but the estimation $S_p = 0.9M_p$ would also be accurate for another implementation). We obtain a total cost of $\approx 256\ \mathrm{Dbl}_{E(\mathbb{F}_p)} + 86\ \mathrm{Add}_{E(\mathbb{F}_p)} \approx\ 2738M_p$ for a scalar multiplication on $\mathbb{G}_1$, 2.4 times this cost: $\approx\ 6590M_p$ for a scalar multiplication in $\mathbb{G}_2$ and $\approx 256S_{p^{12}} + 86M_{p^{12}} \approx 9252M_p$ for an exponentiation in $\mathbb{G}_T$. We note that checking that an element is in $\mathbb{G}_T$ is much more expensive than performing an exponentiation in $\mathbb{G}_T$. Indeed $\mathbb{G}_T$ is an order-r subgroup in a large finite field $\mathbb{F}_{p^k}$. $\mathbb{G}_T$ has particular properties permitting very fast squaring that $\mathbb{F}_{p^k}$ does not. We summarize these estimates in Table 1 (which may be of independent interest).

Table 1. Estimations for common operations in algorithms, for a BN curve with $\log p = 256$ bits and Relic [1] implementation (Running Relic toolkit on a Intel Xeon E5-1603 at 2.80 GHz).

Operation	Cost	Total over $\mathbb{F}_p$	Relic
$\mathbb{F}_{p^k}$ arithmetic			
		M_p	0.149 µs
M_{p^2}	$3M_p$	$3M_p$	0.427 µs
S_{p^2}	$2M_p$	$2M_p$	0.360 µs
M_{p^6}	$6M_{p^2}$	$18M_p$	3.362 µs
S_{p^6}	$2M_{p^2} + 3S_{p^2}$	$12M_p$	2.523 µs
$M_{p^{12}}$	$3M_{p^6}$	$54M_p$	10.856 µs
$S_{p^{12}}$	$2M_{p^6}$	$36M_p$	7.598 µs
$S_{\phi_{12}(p)}$	z^2, $z \in \mathbb{F}_{p^{12}}$, $\mathrm{Norm}(z) = 1$	$18M_p$	4.731 µs
z^a, for any z, a	$\log a\ S_{p^{12}} + \log a\ /3\ M_{p^{12}}$	$54 \log a\ M_p$	3.864 ms
z^a, $\mathrm{Norm}_{\mathbb{F}_{p^{12}}/\mathbb{F}_p}(z) = 1$	$\log a\ S_{\phi_{12}(p)} + \log a\ /3\ M_{p^{12}}$	$36 \log a\ M_p$	2.818 ms
$\mathrm{Norm}_{\mathbb{F}_{p^{12}}/\mathbb{F}_p}(z)$, for any z	$\mathrm{Norm}_{\mathbb{F}_{p^{12}}/\mathbb{F}_{p^6}/\mathbb{F}_{p^2}/\mathbb{F}_p}(z)$	$59\ M_p$	–
z^r, $\mathrm{Norm}_{\mathbb{F}_{p^{12}}/\mathbb{F}_p}(z) = 1$	$z^p z^{1-t} = z^p (z^{p^6})^{t-1}$	$4616\ M_p$	–
check order$(z) = r$ in $\mathbb{F}_{p^k}$	$\mathrm{Norm}_{\mathbb{F}_{p^{12}}/\mathbb{F}_p}(z) = 1$; $z^r = 1$	$4675\ M_p$	–
$E(\mathbb{F}_p)$ arithmetic			
Doubling (Dbl_p)	$2M_p + 5S_p$	$7M_p$	1.043 µs
Addition (Add_p)	$7M_p + 4S_p$	$11M_p$	1.639 µs
Scalar mult. $[a]P$	$\log a\ \mathrm{Dbl} + \log a\ /3\,\mathrm{Add}$	$10.7 \log a\ M_p$	–
$[a_1]P_1 + [a_2]P_2$	$\max(\log a_1, \log a_2)$ $(\mathrm{Dbl} + 2/3\,\mathrm{Add})$	$\max(\log a_1, \log a_2)$ $14.33M_p$	–
$E(\mathbb{F}_{p^2})$ arithmetic			
Doubling (Dbl_{p^2})	$2M_{p^2} + 5S_{p^2}$	$16M_p$	3.137 µs
Addition (Add_{p^2})	$7M_{p^2} + 4S_{p^2}$	$29M_p$	4.866 µs
Scalar mult. $[b]Q$	$\log b\ \mathrm{Dbl}_{p^2} + \log b\ /3\,\mathrm{Add}_{p^2}$	$25.7 \log b\ M_p$	2.017 ms
$[b_1]Q_1 + [b_2]Q_2$	$\max(\log b_1, \log b_2)$ $(\mathrm{Dbl}_{p^2} + 2/3\,\mathrm{Add}_{p^2})$	$\max(\log b_1, \log b_2)$ $35.33M_p$	–
Pairing on a BN curve with $\log_2 p = 256$			
Dbl step + $\ell_{T,T}(P)$	$3M_{p^2} + 7S_{p^2} + 4M_p$	$27\ M_p$	6.036 µs
Add step + $\ell_{T,Q}(P)$	$11M_{p^2} + 2S_{p^2} + 4M_p$	$41\ M_p$	7.593 µs
Miller loop	see Algorithm 1	$8425M_p$	1.776 ms
Final powering	see Algorithm 1	$7911M_p$	1.465 ms
Pairing	see Algorithm 1	$16336M_p$	3.241 ms

Algorithm 1. Optimal Ate Pairing $e_{\mathrm{OptAte}}(P, Q)$ on a BN curve

```
Input: $E(\mathbb{F}_p): y^2 = x^3 + b$, $P(x_P, y_P) \in E(\mathbb{F}_p)[r]$, $Q(x_Q, y_Q) \in E'(\mathbb{F}_{p^2})[r]$, $t$
       trace, $x$ curve parameter
Output: $e_{\mathrm{OptAte}}(P, Q) \in \mathbb{G}_T \subset \mathbb{F}_{p^{12}}^*$
1  $R(X_R : Y_R : Z_R) \leftarrow (x_Q : y_Q : 1)$
2  $f \leftarrow 1$
3  $s \leftarrow 6x + 2$
4  for $m \leftarrow \lfloor \log_2(s) \rfloor - 1, \ldots, 0$ do
5      $R \leftarrow [2]R$; $\ell \leftarrow \ell_{R,R}(P)$                 $3M_{p^2} + 7S_{p^2} + 4M_p = 27M_p$
6      $f \leftarrow f^2 \cdot \ell$                                         $S_{p^{12}} + 13M_{p^2} = 36 + 39 = 75M_p$
7      if $s_m = 1$ or $s_m = -1$ then
8          $R \leftarrow R \pm Q$; $\ell \leftarrow \ell_{R,\pm Q}(P)$       $11M_{p^2} + 2S_{p^2} + 4M_p = 41M_p$
9          $f \leftarrow f \cdot \ell$                                       $13M_{p^2} = 39M_p$

                         total Miller function: $\mathbf{\log s \cdot 102M_p + \log s/3 \cdot 80M_p}$
                   Miller function (e.g. $\log_2 s = 64$): $6528 + 1760 = 8288M_p$
10 $Q_1 \leftarrow \pi_p(Q)$                                                 $M_{p^2} = 3M_p$
11 $R \leftarrow R + Q_1$; $\ell \leftarrow \ell_{R,Q_1}(P)$                $3M_{p^2} + 7S_{p^2} + 4M_p = 27M_p$
12 $f \leftarrow f \cdot \ell$                                               $13M_{p^2} = 39M_p$
13 $Q_2 \leftarrow \pi_{p^2}(Q)$                                             $2M_p$
14 $R \leftarrow R - Q_2$; $\ell \leftarrow \ell_{R,-Q_2}(P)$               $3M_{p^2} + 7S_{p^2} + 4M_p = 27M_p$
15 $f \leftarrow f \cdot \ell$                                               $13M_{p^2} = 39M_p$
                                                                  total: $137M_p$
                              total Miller Loop: $\mathbf{137 + 8288 = 8425M_p}$
16 $f \leftarrow f^{p^6-1}$     $3M_{p^6} + 2S_{p^6} + 10M_{p^2} + 3S_{p^2} + 2M_p + 2S_p + I_p = 118M_p + I_p$
17 $f \leftarrow f^{p^2+1}$                                                  $10M_p + M_{p^{12}} = 64M_p$
18 if $x < 0$ then
19     $a \leftarrow f^{6|x|-5}$
20 else $(f^{p^6} = f^{-1})$
21     $a \leftarrow (f^{p^6})^{6x+5}$
       $\frac{\log p}{4} S_{\Phi_6(p^2)} + \frac{\log p}{12} M_{p^{12}} = 64 \cdot 18 + 22 \cdot 54M_p = (1152 + 1188)M_p = 2340M_p$
22 $b \leftarrow a^p$                                                        $5M_{p^2} = 15M_p$
23 $b \leftarrow ab$                                                         $M_{p^{12}} = 54M_p$
24 Compute $f^p$, $f^{p^2}$ and $f^{p^3}$                     $5M_{p^2} + 10M_p + 5M_{p^2} = 40M_p$
25 $c \leftarrow b \cdot (f^p)^2 \cdot f^{p^2}$                              $S_{\Phi_6(p^2)} + 2M_{p^{12}} = 126M_p$
26 $c \leftarrow c^{6x^2+1}$
   $\frac{\log p}{2} S_{\Phi_6(p^2)} + \frac{\log p}{6} M_{p^{12}} = 128 \cdot 18 + 43 \cdot 54M_p = 2304 + 2322 = 4626M_p$
27 $f \leftarrow f^{p^3} \cdot c \cdot b \cdot (f^p \cdot f)^9 \cdot a \cdot f^4$   $7M_{p^{12}} + 5S_{\Phi_6(p^2)} = 468M_p$
          Exponentiation $f \leftarrow f^{(p^6-1)(p^2+1)(p^4-p^2+1)/r}$: $7851M_p + I_p \approx \mathbf{7911M_p}$
28 return $f$                                                     Pairing: $\mathbf{16336M_p}$
```

Optimal Ate Pairing on a Barreto–Naehrig Curve. A pairing is computed in two steps (see Algorithm 1): a Miller function $f \leftarrow f_{r,Q}(P)$ (Algorithm 1, lines 1–15) followed by a final powering $f^{\frac{p^k-1}{r}}$ (Algorithm 1, lines 16–27) to obtain a unique value in $\mathbb{G}_T$, the subgroup of order r of $\mathbb{F}_{p^k}^*$.

There are several papers on pairing computation on BN curves. We present in Algorithm 1 all the steps for an optimal ate pairing computation on a BN curve. Our global estimate is 16 336 M_p (multiplications in $\mathbb{F}_p$) for one pairing. The Miller loop takes 8425 M_p (52 %) and the exponentation 7911 M_p (48 %). From Relic benchmarks on an Intel Xeon CPU E5-1603 0 at 2.8 GHz, we obtain one pairing in 3.241 ms, the Miller loop in 1.776 ms (55 %) and the exponentiation in 1.465 ms (45 %).

Security Model for Pairing Delegation. In this subsection, we provide an informal description of the security model for pairing delegation protocol and refer the reader to the papers [7,9] for more details. We consider only protocols for delegation of a pairing $e(\mathcal{SK},\mathcal{PP})$ of a secret key $\mathcal{SK}$ and some public parameter $\mathcal{PP}$. The security notions defined in [7,9] are the following:

Secrecy requires that the powerful device cannot learn any information on $\mathcal{SK}$.
Verifiability requires that the restricted device, even interacting with a dishonest powerful device, will not output a wrong value for $e(\mathcal{SK},\mathcal{PP})$.

The formal security game for secrecy is similar to the indistinguishability security notion for encryption schemes. The adversary chooses two secret points $\mathcal{SK}_0$ and $\mathcal{SK}_1$ and runs the delegation protocol with the restricted device for the secret point $\mathcal{SK}_b$ for some bit b. The scheme achieves secrecy if the probability that a (polynomial-time) adversary guesses the bit b is negligibly close to 1/2. The formal security game for verifiability ensures that at the end of the delegation protocol, the restricted device obtains the actual value $e(\mathcal{SK},\mathcal{PP})$ or knows that the powerful device cheated in the protocol.

As mentioned above, in some cases, the secrecy property is too strong if the powerful device is allowed to learn the value $e(\mathcal{SK},\mathcal{PP})$ afterwards. Indeed this value reveals some information on $\mathcal{SK}$ and the complete protocol does not achieve the secrecy property. Therefore, we propose the following notion which is weaker than the secrecy notion but well-suited for our usecases of Pay-TV and encrypted GSM communication:

Weak Secrecy requires that the powerful device cannot learn any information about $\mathcal{SK}$ except what can be deduced from the value $e(\mathcal{SK},\mathcal{PP})$.

Let us assume that we use a pairing delegation protocol for the decryption in a pairing-based scheme (such as the well-known Boneh-Franklin identity-based encryption [4]). If the delegation protocol achieves only Weak Secrecy, a malicious powerful device can mount a lunch-time attack (or CCA1) against the encryption scheme (using the restricted device in the delegation protocol as a decryption oracle). However, since it does not learn any information about $\mathcal{SK}$ (except from the one-time session keys $e(\mathcal{SK},\mathcal{PP}_i)$ for several public parameters $\mathcal{PP}_i$'s), it is not able to decrypt any ciphertext if the restricted device is no longer active (e.g. after revocation of the decryption rights in the Pay-TV scenario).

3 Review of Previous Proposals

3.1 Girault-Lefranc Pairing Delegation Protocol

In this subsection, we present Girault-Lefranc protocol for server-aided signature verification [12, Sect. 4.1] in Algorithm 2 with a performance estimation on a BN curve at a 128-bit security level ($\log r = \log p = 256$) using the Relic library described above.

Our cost estimation concludes that the delegation of $e(\mathcal{SK}, \mathcal{SP})$ with secret $\mathcal{SK}, \mathcal{SP}$ costs $\approx 18640 M_p$ which is more than a pairing computation at the state of the art (we estimate this for $16336 M_p$ in Relic library).

Note that if pre-computation is possible, then the computation of $[a]\mathcal{SK}$ in the first step of Algorithm 2 can actually be done *off-line*. If moreover, the point $\mathcal{SP}$ is public, then the complexity of the delegation protocol falls down to 9252 M_p (i.e. 0.6 pairing). This basic scheme (with pre-computation) is the most efficient pairing delegation protocol (without verifiability) of a pairing $e(\mathcal{SK}, \mathcal{PP})$ of a secret key $\mathcal{SK}$ and some public parameter $\mathcal{PP}$.

In Girault-Lefranc delegation, as f is a pairing output, we can use the optimized squaring formula of Granger and Scott [13] when computing $f^{(ab)^{-1}}$, hence $S_{p^{12}} = 18M_p$ instead of $36M_p$. The computations over the group $\mathbb{G}_1$ might be available on the restricted device such as a smartcard. More precisely, we need multiplication (M_p), addition - subtraction (A_p) and inversion I_p in $\mathbb{F}_p$. Finite field operations are implemented on a smartcard e.g. for ECDSA but the arithmetic operations are not available for the user. We can use the RSA primitives to simulate $\mathbb{F}_p$ arithmetic. We set no padding, the exponent to 2 and the "RSA modulus" to p to get squares mod p, then simulate multiplications through $2xy = (x+y)^2 - x^2 - y^2$. Computations in the group $\mathbb{G}_T$ are not available and must be implemented. If a BN curve [3] is used, $\mathbb{G}_T \subset \mathbb{F}_{p^{12}}^*$ hence a complicated arithmetic must be implemented.

Remark 1 (Lightening the Girault-Lefranc scheme). If the session key $\mathcal{K}$ can be known by the untrusted helper (*i.e.* if one only needs weak secrecy), we

Algorithm 2. Girault-Lefranc Secure pairing delegation [12].

```
  Input: secret points $\mathcal{SK} \in \mathbb{G}_1$ and $\mathcal{SP} \in \mathbb{G}_2$ of prime order $r$, elliptic curve parameters
  Output: corresponding session key $\mathcal{K} = e(\mathcal{SK}, \mathcal{SP})$
1 Sample random $a, b \in \mathbb{Z}_r$ and compute $I = [a]\mathcal{SK}$, $J = [b]\mathcal{SP}$.
  $[a]\mathcal{SK}$ on $E(\mathbb{F}_p)$: $\approx 256$ Dbl$_{E(\mathbb{F}_p)}$ $+ 86$ Add$_{E(\mathbb{F}_p)}$ $\approx 256 \cdot (2M_p + 5S_p) + 86 \cdot (7M_p + 4S_p)$
  $\approx 2738 M_p$
  $[b]\mathcal{SP}$ on $E'(\mathbb{F}_{p^2})$: $\approx 256$Dbl$_{E'(\mathbb{F}_{p^2})}$ $+ 86$ Add$_{E'(\mathbb{F}_{p^2})}$        $\approx 6590 M_p$
  If $\mathcal{SP}$ is public we can set $b = 1$
2 Send $I, J$ to the server.
3 Compute $(ab)^{-1}$ mod $r$.                                   $\approx 60 M_p$
4 Receive $f = e(I, J)$.                          delegated: $\approx 16336 M_p$
5 Compute $f^{(ab)^{-1}}$ to retrieve $\mathcal{K} = e(\mathcal{SK}, \mathcal{SP})$.          $\approx 9252 M_p$
6 return $\mathcal{K}$.      Total cost ($b = 1$): $\approx 9252 + 60 + 2738 \approx 12050 M_p = 0.74$ pairing
                      Total cost $a, b \neq 1$: $\approx 12050 + 6590 \approx 18640 M_p = 1.14$ pairing
```

note that a variant of the protocol may be used in some cases. We propose to ask the external resource to compute $e([\alpha]\mathcal{SK}, [\alpha^{-1}]\mathcal{SP}) = e(\mathcal{SK}, \mathcal{SP}) = \mathcal{K}$ with α taken at random. The output will be exactly $\mathcal{K}$. This solution is not very efficient as it costs $9388/16336 = 0.6$ pairing. To improve it slightly in practice, we can swap $\mathcal{SK}$ and $\mathcal{PP}$, i.e. put $\mathcal{SK}$ in $E^{'}(\mathbb{F}_{p^2})$ and $\mathcal{PP} \in E(\mathbb{F}_p)$. In this way, $[\alpha]\mathcal{SK}$ is the costly part and can be computed offline. Note that this delegation procedure reveals some information on the secret key $\mathcal{SK}$ and it is necessary to reprove the security of the underlying scheme if it is used to improve its efficiency.

3.2 Chevallier-Mames *et al.* Pairing Delegation Protocol

Another pairing delegation protocol was introduced by Chevallier-Mames, Coron, McCullagh, Naccache and Scott in 2005 [8,9]. Contrary to Girault-Lefranc's protocol, the protocol proposed by Chevallier-Mames *et al.* achieves secrecy (unconditionnally) and verifiability. Unfortunately, the protocol is very inefficient since the overall cost for the restricted device is 3.5 times the cost for computing the pairing (3.3 if pre-computation is possible). The main advantage

Algorithm 3. Pairing delegation with public right-side point [7, Sect. 4.1].

```
   Input: secret point $\mathcal{SK} \in \mathbb{G}_1$ and public point $\mathcal{PP} \in \mathbb{G}_2$ of prime order $r$, $G_1$
          generator of $\mathbb{G}_1$, $G_2$ of $\mathbb{G}_2$, elliptic curve parameters
   Output: Pairing value $e(\mathcal{SK}, \mathcal{PP})$
 1 Sample a random $a \in \mathbb{Z}_r$ and compute $I_1 = [a]G_1$.   $[a]G_1$ on $E(\mathbb{F}_p)$: $\approx 2738 M_p$
 2 Sample a random $b \in \mathbb{Z}_r$ and compute $I_2 = [b]G_2$.   $[b]G_1$ on $E(\mathbb{F}_p)$: $\approx 2738 M_p$
 3 Compute $\chi = e(G_1, G_2)^{ab}$                                 1 exp. in $\mathbb{G}_T$ $\approx 9216 M_p$
 4 Compute $(a)^{-1} \mod r$ and $(b)^{-1} \mod r$                   $I_p + 3M_p \approx 63 M_p$
 5 Sample c random $c \in \mathbb{Z}_r$ and compute $J_0 = [c]\mathcal{SK}$.   $[c]\mathcal{SK}$ on $E(\mathbb{F}_p)$: $\approx 2738 M_p$
 6 Compute $J_1 = [b^{-1}]J_0 + I_1$.                                $[b^{-1}]J_0$ on $E(\mathbb{F}_p)$: $\approx 2738 M_p$
 7 Compute $J_2 = [a^{-1}]\mathcal{PP} + I_2$.                        $[a^{-1}]\mathcal{PP}$ on $E(\mathbb{F}_p)$: $\approx 2738 M_p$
 8 Send $J_1, J_2, \mathcal{PP}$ to the server.
 9 Ask for $\alpha_1 = e(J_1, J_2)(e(G_1, \mathcal{PP})e(J_0, G_2))^{-1}, \alpha_2 = e(J_0, \mathcal{PP})$   delegated:
   $\approx 4 \cdot 16336 M_p = 65344 M_p$
10 Receive $\alpha_1, \alpha_2$
11 Check that $\alpha_2 \in \mathbb{G}_T$: compute $\alpha_2^r$                       $4675 M_p$
12 if $\alpha_2^r \neq 1$ then
13     outputs $\perp$ and halt.
14 Compute $\chi' = \chi \cdot \alpha_2^{(ab)^{-1}}$                          1 exp. in $\mathbb{G}_T$ $\approx 9216 M_p$
15 if $\chi' = \alpha_1$ then
16     compute $(c)^{-1} \mod r$                                      $I_p \approx 60 M_p$
17     outputs $\alpha_2^{(c)^{-1}}$ and halt.                          1 exp. in $\mathbb{G}_T$ $\approx 9216 M_p$
18 else
19     outputs $\perp$ and halt.
                                              Total cost: $46136 M_p = 2.8$ pairings
                              Cost w/o pre-computation: $25905 M_p = 1.6$ pairings
```

of the scheme is to save of area that is required to implement a pairing in the restricted device such a smart card. However, as mentioned above, even if we can use tricks, computations in the group $\mathbb{G}_T$ are usually not available and must be implemented (i.e. complex arithmetic in $\mathbb{G}_T \subset \mathbb{F}_{p^{12}}^*$ for a BN curve).

3.3 Canard-Devigne-Sanders Pairing Delegation Protocol

We present in Algorithm 3 the pairing delegation protocol proposed recently by Canard, Devigne and Sanders [7]. The protocol is more efficient than the previous one. It also achieves secrecy (unconditionnally) and verifiability. Canard *et al.* actually showed that their proposal is in fact more efficient than the computation of the pairing for optimal ate pairing on a so-called KSS-18 curve [15]. Unfortunately, as shown by the precise complexity of Algorithm 3, this is not the case for state-of-the-art optimal Ate pairing on a BN curve [3]. More precisely, we show that the overall cost for the restricted device is 2.8 times the cost for computing the pairing (1.6 if pre-computation is possible).

4 Pairing Delegation with Knapsack

We present in this section a new approach to perform pairing delegation (without verifiability) of a pairing $e(\mathcal{SK}, \mathcal{PP})$ of a secret key $\mathcal{SK}$ and some public parameter $\mathcal{PP}$. The restricted device (e.g. a smartcard) generates random points and sends them to the powerful device to compute several pairings. The smartcard receives the pairings and combines some of them to get the actual value $e(\mathcal{SK}, \mathcal{PP})$. The basic idea is to mask the secret value $\mathcal{SK}$ by a linear combination of those random points with "small" coefficients to improve efficiency. A similar approach has been used successfully in the setting of server-aided exponentiation [6,16].

4.1 Security Analysis

Let $\mathbb{G}$ be a cyclic group of order p denoted additively. We consider the two following distributions:

$$\mathcal{U}_n = \{(P_1, P_2, \ldots, P_n, Q) \overset{R}{\longleftarrow} \mathbb{G}^{n+1}\}$$

and

$$\mathcal{K}_{n,A} = \left\{ (P_1, P_2, \ldots, P_n, Q), \text{ s.t. } \begin{array}{r} (P_1, P_2, \ldots, P_n) \overset{R}{\longleftarrow} \mathbb{G}^n \\ Q \leftarrow [a_1]P_1 + \cdots + [a_n]P_n \\ \text{where } (a_1, \ldots, a_n) \overset{R}{\longleftarrow} [\![0, A-1]\!]^n \end{array} \right\}.$$

$\mathcal{U}_n$ is the uniform distribution on $\mathbb{G}^{n+1}$ and $\mathcal{K}_{n,A}$ outputs $(n{+}1)$-tuples where the first n components are picked uniformly at random in $\mathbb{G}$ while the last component is a linear combination of those elements with exponents picked uniformly at random in the interval $[\![0, A-1]\!]$. In a basic version of our delegation protocol, the restricted device sends the elements $(P_1, \ldots, P_n)$ and $P_{n+1} = (\mathcal{SK} - Q)$ to the powerful device. It replies by sending back the pairings $e(P_i, \mathcal{PP})$ for $i \in \{1, \ldots, n+1\}$. The restricted device finally gets $e(\mathcal{SK}, \mathcal{PP})$ as $e(P_{n+1}, \mathcal{PP}) \cdot \prod_{i=1}^{n} e(g_i, \mathcal{PP})^{a_i}$. If the two distributions $\mathcal{U}_n$ and $\mathcal{K}_{n,A}$ are indistinguishable, the protocol will readily achieve the secrecy property.

- *Perfect indistinguishability.* It is straightforward to see that if $A = p$, then the two distributions are identical (even if $n = 1$) and the delegation scheme as outlined above achieves unconditional secrecy. Unfortunately, as we will see in the next paragraph, the efficiency of our schemes depends crucially on the size of A and one wants to use smaller A in practice.
- *Statistical indistinguishability.* By using classical results on the distribution of modular sums [16], one can prove that if $A^n = \Omega(p^2)$, then the two distributions $\mathcal{U}_n$ and $\mathcal{K}_{n,A}$ are statistically indistinguishable (see [10,16] for details). For these parameters, the delegation protocol achieves statistical (and therefore computational) secrecy. For cryptographic purposes, the order p of $\mathbb{G}$ needs to be of $2k$-bit size to achieve a k-bit security level. Therefore, to achieve statistical indistinguishability, we need to have $A^n = \Omega(2^{4k})$ and the resulting delegation protocol is not really efficient.
- *Computational indistinguishability.* For smaller parameters (i.e. $A^n = o(p^2)$), we cannot prove that the $\mathcal{U}_n$ and $\mathcal{K}_{n,A}$ are statistically indistinguishable. However, it may be possible to prove that they are computationally indistinguishable. Using a variant of Shanks "baby-step giant-step" algorithm, one can see easily that it is possible to find the scalars $(a_1, \ldots, a_n)$ (if they exist) such that $Q = [a_1]P_1 + \cdots + [a_n]P_n$ in $O(A^{n/2})$ group operations in $\mathbb{G}$ (i.e. to solve the *generalized knapsack problem* in $\mathbb{G}$). Therefore, to achieve computational indistinguishability for a k-bit security parameter, one needs to have at least $A^n = \Omega(2^{2k}) = \Omega(p)$.

To conclude this paragraph, we will prove that the two distributions $\mathcal{U}_n$ and $\mathcal{K}_{n,A}$ are computationally indistinguishable in the generic group model when $A^n = \Omega(2^{2k}) = \Omega(p)$. Our delegation protocol therefore achieves secrecy in the generic group model when $A^n = \Omega(2^{2k}) = \Omega(p)$. This model was introduced by Shoup [17] for measuring the exact difficulty of solving discrete logarithm problems. Algorithms in generic groups do not exploit any properties of the encodings of group elements. They can access group elements only via a random encoding algorithm that encodes group elements as random bit-strings.

Let $\mathcal{A}$ be a generic group adversary. As usual, the generic group model is implemented by choosing a random encoding $\sigma : \mathbb{G} \longrightarrow \{0,1\}^m$. Instead of working directly with group elements, $\mathcal{A}$ takes as input their image under σ. This way, all $\mathcal{A}$ can test is string equality. $\mathcal{A}$ is also given access to an oracle computing group addition and subtraction: taking $\sigma(R_1)$ and $\sigma(R_2)$ and returning $\sigma(R_1 + R_2)$, similarly for subtraction. Finally, we can assume that $\mathcal{A}$ submits to the oracle only encodings of elements it had previously received. This is because we can choose m large enough so that the probability of choosing a string that is also in the image of σ is negligible.

Theorem 1. *Let $\mathcal{A}$ be a generic algorithm that distinguishes the two distributions $\mathcal{U}_n$ and $\mathcal{K}_{n,A}$ that makes at most τ group oracle queries, then $\mathcal{A}$'s advantage in distinguishing the two distributions is upper-bounded by $O(\tau^2/A^n)$.*

To prove this theorem, we consider the following distributions in a product group $\mathbb{G}_1 \times \cdots \times \mathbb{G}_n$ where each $\mathbb{G}_i$ is cyclic group of prime order p (for $i \in \{1, \dots, n\}$).

$$\mathcal{U}'_n = \{(P_1, P_2, \dots, P_n, Q) \xleftarrow{R} \mathbb{G}_1 \times \mathbb{G}_2 \times \cdots \times \mathbb{G}_n \times (\mathbb{G}_1 \times \mathbb{G}_2 \times \cdots \times \mathbb{G}_n)\}$$

and

$$\mathcal{K}'_{n,A} = \left\{ (P_1, P_2, \dots, P_n, Q), \text{ s.t. } \begin{array}{r} (P_1, P_2, \dots, P_n) \xleftarrow{R} \mathbb{G}_1 \times \mathbb{G}_2 \times \cdots \times \mathbb{G}_n \\ Q \leftarrow [a_1]P_1 + \cdots + [a_n]P_n \qquad\qquad \\ \text{where } (a_1, \dots, a_n) \xleftarrow{R} [\![0, A-1]\!]^n \end{array} \right\}$$

It is worth mentioning that the use of these product groups in cryptography is not interesting since even if their order is p^n, the complexity of discrete logarithm computation in them is not much harder than in cyclic groups of order p. We will only use them as a tool in order to prove our Theorem 1.

Following Shoup's technique [17], it is easy to prove that a generic algorithm in the product group $\mathbb{G}_1 \times \cdots \times \mathbb{G}_n$ (or equivalently in $\mathbb{Z}_p^n$) has a negligible advantage in distinguishing the two distributions $\mathcal{U}'_n$ and $\mathcal{K}'_{n,A}$ if it makes a polynomial number of group oracle queries. More precisely, we can prove the following proposition:

Proposition 1. *Let $\mathcal{A}$ be a generic algorithm that distinguishes the two distributions $\mathcal{U}'_n$ and $\mathcal{K}'_{n,A}$ and makes at most τ group oracle queries, then $\mathcal{A}$'s advantage in distinguishing the two distributions is upper-bounded by $O(\tau^2/A^n)$.*

Proof. We consider an algorithm $\mathcal{B}$ playing the following game with $\mathcal{A}$. Algorithm $\mathcal{B}$ chooses $n+1$ bit strings $\sigma_1, \dots, \sigma_n, \sigma_{n+1}$ uniformly in $\{0,1\}^m$. Internally, $\mathcal{B}$ keeps track of the encoded elements using elements in the ring $\mathbb{Z}_p[X_1] \times \cdots \times \mathbb{Z}_p[X_n]$. To maintain consistency with the bit strings given to $\mathcal{A}$, $\mathcal{B}$ creates a lists $\mathcal{L}$ of pairs (F, σ) where F is a polynomial vector in the ring $\mathbb{Z}_p[X_1] \times \cdots \times \mathbb{Z}_p[X_n]$ and $\sigma \in \{0,1\}^m$ is the encoding of a group element. The polynomial vector F represents the exponent of the encoded element in the group $\mathbb{G}_1 \times \cdots \times \mathbb{G}_n$. Initially, $\mathcal{L}$ is set to

$$\{((1,0,\dots,0),\sigma_1),((0,1,\dots,0),\sigma_2),\dots,((0,0,\dots,1),\sigma_n),((X_1,\dots,X_n),\sigma_{n+1})\}$$

Algorithm $\mathcal{B}$ starts the game providing $\mathcal{A}$ with $\sigma_1, \dots, \sigma_n, \sigma_{n+1}$. The simulation of the group operations oracle goes as follows:

Group Operation: Given two encodings σ_i and σ_j in $\mathcal{L}$, $\mathcal{B}$ recovers the corresponding vectors F_i and F_j and computes $F_i + F_j$ (or $F_i - F_j$) termwise. If $F_i + F_j$ (or $F_i - F_j$) is already in $\mathcal{L}$, $\mathcal{B}$ returns to $\mathcal{A}$ the corresponding bit string; otherwise it returns a uniform element $\sigma \xleftarrow{R} \{0,1\}^m$ and stores $(F_i + F_j, \sigma)$ (or $(F_i - F_j, \sigma)$) in $\mathcal{L}$.

After $\mathcal{A}$ queried the oracles, it outputs a bit b. At this point, $\mathcal{B}$ chooses a random bit $b^* \in \{0, 1\}$ and uniform values $x_1, \ldots, x_n \in \mathbb{Z}_p$ if $b^* = 0$ or uniform values $x_1, \ldots, x_n \in [\![0, A-1]\!]$ if $b^* = 1$. The algorithm $\mathcal{B}$ sets $X_i = x_i$ for $i \in \{1, \ldots, n\}$.

If the simulation provided by $\mathcal{B}$ is consistent, it reveals nothing about b. This means that the probability of $\mathcal{A}$ guessing the correct value for b^* is $1/2$. The only way in which the simulation could be inconsistent is if, after we choose value for $x_1, \ldots, x_n$, two different polynomial vectors in $\mathcal{L}$ happen to produce the same value. First, note that $\mathcal{A}$ is unable to cause such a collision on its own. Indeed, notice that $\mathcal{L}$ is initially populated with polynomials of degree at most one in each coordinate and that both the group addition and subtraction oracle do not increase the degree of the polynomial. Thus, all polynomials contained in $\mathcal{L}$ have degree at most one. This is enough to conclude that $\mathcal{A}$ cannot purposely produce a collision.

It remains to prove that the probability of a collision happening due to a unlucky choice of values is negligible. In other words, we have to bound the probability that two distinct F_i, F_j in $\mathcal{L}$ evaluate to the same value after the substitution, namely $F_i(x_1, \ldots, x_n) - F_j(x_1, \ldots, x_n) = 0$. This reduces to bound the probability of hitting a zero of $F_i - F_j$. By the simulation, this happens only if $F_i - F_j$ is a non-constant polynomial vector and in this case, each coordinate is a degree one polynomial in one X_i's.

Recall that the Schwartz-Zippel lemma says that, if F is a degree d polynomial in $\mathbb{Z}_p[X_1, \ldots, X_n]$ and $S \subseteq \mathbb{Z}_p$ then

$$\Pr[F(x_1, \ldots, x_n) = 0] \leq \frac{d}{|S|}$$

where $x_1, \ldots, x_n$ are chosen uniformly from S. Going back to our case, we obtain by applying the Schwartz-Zippel lemma to each coordinate:

$$\Pr[(F_i - F_j)(x_1, \ldots, x_n) = \mathbf{0} \in \mathbb{Z}_p^n] \leq \begin{cases} 1/p^n \text{ if } b^* = 0 \\ 1/A^n \text{ if } b^* = 1 \end{cases}$$

Therefore, the probability that the simulation provided by $\mathcal{B}$ is inconsistent is upper-bounded by $\tau(\tau - 1)/A^n$. □

We will now prove that, provided m is large enough, a generic algorithm is not able to decide whether it is given as inputs n generators $(P_1, \ldots, P_n)$ in a cyclic group $\mathbb{G}$ of prime order p or n order-p elements

$$(P_1, 1_{\mathbb{G}_2}, \ldots, 1_{\mathbb{G}_n}), (1_{\mathbb{G}_1}, P_2, \ldots, 1_{\mathbb{G}_n}), \ldots, (1_{\mathbb{G}_1}, 1_{\mathbb{G}_2}, \ldots, P_n)$$

in a product group $\mathbb{G}_1 \times \cdots \times \mathbb{G}_n$ where each $\mathbb{G}_i$ is cyclic group of prime order p. Note that the groups $\mathbb{G}$ and $\mathbb{G}_1 \times \cdots \times \mathbb{G}_n$ are not of the same order and in practice, it will probably be easy to distinguish them. We only claim that this is difficult for a generic algorithm.

Proposition 2. *Let $\mathcal{A}$ be a generic algorithm that distinguishes these two settings and makes at most τ group oracle queries, then $\mathcal{A}$'s advantage in distinguishing the two distributions is upper-bounded by $O(\tau^2/p)$.*

Proof. We consider an algorithm $\mathcal{B}$ playing the following game with $\mathcal{A}$. Algorithm $\mathcal{B}$ chooses a random bit b^* and runs one of the following simulation depending on the bit b^*

– If $b^* = 0$, $\mathcal{B}$ chooses n bit strings $\sigma_1, \dots, \sigma_n$ uniformly in $\{0,1\}^m$. Internally, $\mathcal{B}$ keeps track of the encoded elements using elements in the ring $\mathbb{Z}_p[X_1, \dots, X_n]$. To maintain consistency with the bit strings given to $\mathcal{A}$, $\mathcal{B}$ creates a list $\mathcal{L}$ of pairs (F, σ) where F is a polynomial in the ring $\mathbb{Z}_p[X_1, \dots, X_n]$ and $\sigma \in \{0,1\}^m$ is the encoding of a group element. The polynomial F represents the exponent of the encoded element in the group $\mathbb{G}$. Initially, $\mathcal{L}$ is set to

$$\{(X_1, \sigma_1), (X_2, \sigma_2), \dots, (X_n, \sigma_n)\}$$

– If $b^* = 1$, $\mathcal{B}$ chooses also n bit strings $\sigma_1, \dots, \sigma_n$ uniformly in $\{0,1\}^m$. Internally, $\mathcal{B}$ keeps track of the encoded elements using elements in the ring $\mathbb{Z}_p[X_1] \times \dots \times \mathbb{Z}_p[X_n]$. To maintain consistency with the bit strings given to $\mathcal{A}$, $\mathcal{B}$ creates a list $\mathcal{L}$ of pairs (F, σ) where F is a polynomial vector in the ring $\mathbb{Z}_p[X_1] \times \dots \times \mathbb{Z}_p[X_n]$ and $\sigma \in \{0,1\}^m$ is the encoding of a group element. The polynomial vector F represents the exponent of the encoded element in the group $\mathbb{G}_1 \times \dots \times \mathbb{G}_n$. Initially, $\mathcal{L}$ is set to

$$\{((X_1, 0, 0, \dots, 0), \sigma_1), ((0, X_2, 0, \dots, 0), \sigma_2), \dots, ((0, 0, \dots, 0, X_n), \sigma_n)\}$$

In each cases, algorithm $\mathcal{B}$ starts the game providing $\mathcal{A}$ with $\sigma_1, \dots, \sigma_n$. The simulation of the group operations oracle goes as follows:

Group operation: Given two encodings σ_i and σ_j in $\mathcal{L}$, $\mathcal{B}$ recovers the corresponding polynomials (or polynomial vectors, depending on b^*) F_i and F_j and computes $F_i + F_j$ (or $F_i - F_j$) termwise. If $F_i + F_j$ (or $F_i - F_j$) is already in $\mathcal{L}$, $\mathcal{B}$ returns to $\mathcal{A}$ the corresponding bit string; otherwise it returns a uniform element $\sigma \xleftarrow{R} \{0,1\}^m$ and stores $(F_i + F_j, \sigma)$ (or $(F_i - F_j, \sigma)$) in $\mathcal{L}$.

After $\mathcal{A}$ queried the oracles, it outputs a bit b. At this point, $\mathcal{B}$ chooses uniform values $x_1, \dots, x_n \in \mathbb{Z}_p$. The algorithm $\mathcal{B}$ sets $X_i = x_i$ for $i \in \{1, \dots, n\}$.

If the simulation provided by $\mathcal{B}$ is consistent, it reveals nothing about b. This means that the probability of $\mathcal{A}$ guessing the correct value for b^* is $1/2$. The only way in which the simulation could be inconsistent is if, after we choose value for $x_1, \dots, x_n$, two different polynomial vectors in $\mathcal{L}$ happen to produce the same value. First, note that $\mathcal{A}$ is unable to cause such a collision on its own. Indeed, notice that $\mathcal{L}$ is initially populated with polynomials of degree at most one in each coordinate and that both the group addition and subtraction oracle do not increase the degree of the polynomial. Thus, all polynomials contained in $\mathcal{L}$ have degree at most one. This is enough to conclude that $\mathcal{A}$ cannot purposely produce a collision.

It remains to prove that the probability of a collision happening due to a unlucky choice of values is negligible. If $b^* = 1$, the probability of a collision happening is equal to 0. If $b^* = 0$, we have to bound the probability that two distinct F_i, F_j in $\mathcal{L}$ evaluate to the same value after the substitution, namely $F_i(x_1, \ldots, x_n) - F_j(x_1, \ldots, x_n) = 0$. This reduces to bound the probability of hitting a zero of $F_i - F_j$.

Applying the Schwartz-Zippel lemma, we obtain

$$\Pr[(F_i - F_j)(x_1, \ldots, x_n) = 0 \in \mathbb{Z}_p] \leq 1/p$$

Therefore, the probability that the simulation provided by $\mathcal{B}$ is inconsistent is upper-bounded by $\tau(\tau - 1)/p$. □

To prove Theorem 1, it is then enough to prove that if there exists a generic algorithm that distinguishes the two distributions $\mathcal{U}_n$ and $\mathcal{K}_{n,A}$ that makes at most τ group oracle queries with an advantage larger than $\Omega(\tau^2/A^n)$, it gives an adversary able to distinguish the cyclic group setting from the product group setting making at most τ group oracle queries and with advantage $\Omega(\tau^2/A^n)$ (due to Proposition 1) and this result contradicts Proposition 2.

4.2 Description of Our Protocol

In the previous subsection, we provided a description of a basic version of our protocol. In this subsection, we consider an improved version of it on elliptic curves equipped with efficient endomorphisms. In this improved scheme, instead of masking $\mathcal{SK}$ with $[a_1]P_1 + \cdots + [a_{n-1}]P_{n-1}$ with $(a_1, \ldots, a_{n-1}) \overset{R}{\longleftarrow} [\![0, A - 1]\!]^{n-1}$, we will mask it with $[a_1]Q_1 + \cdots + [a_{n-1}]Q_{n-1}$ with $(a_1, \ldots, a_{n-1}) \overset{R}{\longleftarrow} [\![0, A - 1\ ]\!]^{n-1}$ where the Q_i's are images of the P_i under one of the efficient endomorphisms defined on the curve. If we denote $\mathcal{S}$ the set of efficient endomorphisms on the curve (that can also be efficiently evaluated in the group $\mathbb{G}_T$), we obtained a scheme with generic security $\Omega(\#\mathcal{S}^{n-1} \cdot A^{n-1/2})$.

Setup (could be offline). In the following, the smartcard has to generate several random points on an elliptic curve $E(\mathbb{F}_p)$. Fouque and Tibouchi [11] proposed an efficient method to do it on a BN curve.

1. Let $\mathcal{I}$ a set of small integers, $\mathcal{I} = \{0, 1, 2, 3, 4, 5, \ldots, 2^\ell - 1\}$ with $\#\mathcal{I} = 2^\ell = A$.
2. The smartcard generates $n-1$ random points $P_1, P_2, \ldots, P_{n-1}$ on the elliptic curve $E(\mathbb{F}_p)$.
3. The smartcard chooses an endomorphism $\sigma_i \in \mathcal{S}$ to apply to P_i and sets $Q_i = \sigma_i(P_i)$.
4. For each point Q_i, the smartcard takes at random $\alpha_i \in \mathcal{I}$ and sets

$$P_n = \mathcal{SK} - ([\alpha_1]Q_1 + [\alpha_2]Q_2 + \ldots + [\alpha_{n-1}]Q_{n-1} = \mathcal{SK} - \sum_{i=1}^{n-1} [\alpha_i]Q_i \ .$$

Delegation

5. The smartcard sends $P_1, P_2, \ldots, P_n$ to the server.
6. The server computes the n pairings $f_i = e(Q_i, \mathcal{PP})$ and sends them back to the smartcard.

Session key computation

7. The smartcard computes $(f_1^{\sigma_1})^{\alpha_1} \cdot (f_2^{\sigma_2})^{\alpha_2} \cdots (f_{n-1}^{\sigma_{n-1}})^{\alpha_{n-1}} \cdot f_n = \mathcal{K}$. The σ_i are also almost free. Thanks to the bilinearity property,

$$\begin{aligned} e(\mathcal{SK}, \mathcal{PP}) &= e(\alpha_1 Q_1 + \alpha_2 Q_2 \ldots + \alpha_{n-1} Q_{n-1} + P_n, \mathcal{PP}) \\ &= e(\alpha_1 Q_1, \mathcal{PP}) e(\alpha_2 Q_2, \mathcal{PP}) \cdots e(\alpha_{n-1} Q_{n-1}, \mathcal{PP}) e(P_n, \mathcal{PP}) \\ &= (e(P_1, \mathcal{PP})^{\sigma_1})^{\alpha_1} \cdots (e(P_{n-1}, \mathcal{PP})^{\sigma_{n-1}})^{\alpha_{n-1}} (e(P_n, \mathcal{PP})) \end{aligned}$$

with σ_i a cheap endomorphism in $\mathbb{F}_{p^k}^*$ such that $e(\sigma_i(P_i), \mathcal{PP}) = e(P_i, \mathcal{PP})^{\sigma_i}$.

Example on a Barreto–Naehrig curve. For optimal Ate pairing on a BN curve with 128-bit security level (i.e. 256-bit prime number p), the endomorphism set $\mathcal{S}$ can be defined as $\{\mathrm{Id}, -\mathrm{Id}, \phi, \phi^2, -\phi, -\phi^2\}$ where ϕ is computed from the complex multiplication endomorphism available on the curve. These endomorphisms are almost free on $E(\mathbb{F}_p)$ if $D = 1$ or $D = 3$. They cost at most one multiplication and one subtraction in $\mathbb{F}_p$ and the resulting point Q_i is still in affine coordinates [5].

In the Setup procedure, the smartcard has to obtain P_n in affine coordinates, this costs one inversion in $\mathbb{F}_p$ plus four multiplications, resulting in an additional cost of (say) $64 M_p$. The cost of computing P_n is $(n-1) \cdot (\ell \cdot 7 + \ell/2 \cdot 11 + 16) + 64 M_p$. Indeed, in Jacobian coordinates, one addition on $E(\mathbb{F}_p)$ with one of the two points in affine coordinates costs $11 M_p$, if none of the points are in affine coordinates, this costs $16 M_p$, and one doubling costs $8 M_p$. If moreover we use a BN curve ($a_4 = 0$), a doubling costs $7 M_p$.

The computation cost for the powerful device is $16336(0.84(n-1)+1)\ M_p$. Indeed, the first pairing costs $\approx 16336 M_p$ and the $(n-1)$ other ones cost 0.84 of one pairing (since the second argument is the fixed point $\mathcal{PP}$, the tangents and lines can be computed from $\mathcal{PP}$ one single time for all the pairings).

Finally, the smartcard computes[1] $n-1$ exponentiations and multiplies n elements in $\mathbb{G}_T$ to obtain the session key $\mathcal{K} = e(\mathcal{SK}, \mathcal{PP})$. An exponentiation costs in average ℓ squaring plus $\ell/2$ multiplications in $\mathbb{F}_{p^{12}}$. The $n-1$ exponentiations cost $(n-1)(18\ell + 54\ell/3) M_p$. It remains to compute $n-1$ multiplications.

Overall, we obtain the global cost for the restricted device is: $(n-1)(73 M_p + 46,7 \ell M_p)$ (and $(n-1)(73 M_p + 36 \ell M_p)$ is pre-computation is possible). We summarize our proposition in Algorithm 4. By choosing appropriate values for n and ℓ, one can achieve various communication-efficiency trade-off as shown in Table 2. To achieve statistical security (instead of generic computational security), one basically needs to double the value of ℓ. One can find parameters for which the delegation procedure is more efficient than the pairing computation (0.5 pairing for practical parameters).

[1] It is worth mentioning that this computational cost can be further decreased by using classical multi-exponentiation techniques (in particular for small values of n (e.g. $n = 5$).

Algorithm 4. Pairing delegation with knapsack.

```
   Input: secret key $\mathcal{SK}$, public value $\mathcal{PP}$, set $I$ of small integers with $\#I = 2^\ell$
   Output: Session key $\mathcal{K} = e(\mathcal{SK}, \mathcal{PP})$
1  Offline:
2  Generate $n-1$ random points $P_1, P_2, \ldots, P_n \in E(\mathbb{F}_p)$.
3  foreach $P_i$ do
4      Choose at random an endomorphism $\sigma_i \in \{\mathrm{Id}, -\mathrm{Id}, \phi, -\phi, \phi^2, -\phi^2\}$      $\sigma_i$ on $E(\mathbb{F}_p)$: at most $1M_p$
5      Choose at random an integer $\alpha_i \in \mathcal{I}$
6      Compute $Q_i = [\alpha_i]\sigma(P_i)$      $[\alpha_i]$: $\log_2 \alpha_i(\mathtt{Dbl}_{E(\mathbb{F}_p)} + \frac{1}{3}\mathtt{Add}_{E(\mathbb{F}_p)}) \leqslant 10.7\ell M_p$
7  Online:
8  Compute $P_n = \mathcal{SK} - ([\alpha_1]\sigma_i(P_1) + [\alpha_2]\sigma_2(P_2) + \ldots + [\alpha_{n-1}]\sigma_{n-1}(P_{n-1}) = \mathcal{SK} - \sum_{i=1}^{n-1}[\alpha_i]\sigma_i(P_i)$
                                                        $n-1$ $\mathtt{Add}_{E(\mathbb{F}_p)} = (n-1)11M_p$
9  Send $\mathcal{PP}$ and all the $P_1, \ldots, P_n$ to the server.   communication: $\log(p) \cdot (n+1)$ bits
10 Ask for all the $f_i = e(P_i, \mathcal{PP}), 1 \leqslant i \leqslant n$            Delegated: $\approx 16336n$ $M_p$
11 Compute $\mathcal{K} = (f_1^{\sigma_1})^{\alpha_1} \cdot (f_2^{\sigma_2})^{\alpha_2} \cdots (f_{n-1}^{\sigma_{n-1}})^{\alpha_{n-1}} \cdot f_n$   $(n-1)(\sigma_i + \alpha_i + \mathtt{Mult.}) = (n-1)(8M_p + \ell(S_{\Phi_{12}(p)} + \frac{1}{3}M_{p^{12}}) + M_{p^{12}}) = (n-1)(62M_p + 36\ell M_p)$
12 return $\mathcal{K}$.                                Total cost: $(n-1)(73M_p + 46,7\ell M_p)$
                                     Cost w/o pre-computation: $(n-1)(73M_p + 36\ell M_p)$
                                           for $n = 20$ and $\ell = 8$: $6859M_p = 0.4$ pairing
```

5 Partial Pairing Computation Delegation

In this final section, we propose a completely different approach based on the arithmetic of the pairing computation (without verifiability) of a pairing $e(\mathcal{SK}, \mathcal{PP})$ of a secret key $\mathcal{SK}$ and some public parameter $\mathcal{PP}$. More precisely, we delegate only the non-critical steps in the pairing algorithm, looking carefully at each instruction in Miller algorithm. The protocol only achieves *weak secrecy*: the helper will learn the session key $\mathcal{K}$ (but still not the secret key $\mathcal{SK}$).

Final Powering Delegation. We can blind the output $f' \leftarrow u \cdot f$ of the Miller function by an element $u \in \mathbb{F}_{p^k}^*$ which is an r-th power (there exists a $u' \in \mathbb{F}_{p^k}^*$ such that $u'^r = u$), see Algorithm 5. Hence u will disappear after the final powering $f^{\frac{p^k-1}{r}}$ (Algorithm 1, lines 16–27) since $u^{\frac{p^k-1}{r}} = u'^{p^k-1} = 1$. So we can delegate the final powering thanks to the equality $f'^{(p^k-1)/r} = (uf)^{(p^k-1)/r} = \mathcal{K}$ the session key. The helper learns the session key $\mathcal{K}$ but has no additional information on f (in particular pairing inversion is not possible).

Tangent and line Delegation. The two points P, Q play two different roles in a Tate-like pairing computation. In an ate pairing, the point P is used to evaluate the intermediate line functions $\ell(P)$. The line functions ℓ are computed through a scalar multiplication $[s]Q$ (with s a public parameter of the curve). The coefficients arising in the lines and tangent computation are re-used to update the

Table 2. Communication/Efficiency Trade-off of our knapsack delegation protocol

ℓ	n	Generic security	Computational cost	Communication
59	5	128	$8788M_p = 0.53$ pairing	15360 bits
23	10	126	$8109M_p = 0.49$ pairing	30720 bits
13	15	127	$7574M_p = 0.46$ pairing	46080 bits
8	20	125	$6859M_p = 0.41$ pairing	61440 bits
8	20	125	$6859M_p = 0.41$ pairing	61440 bits
5	25	122	$6072M_p = 0.37$ pairing	76800 bits
3	30	118	$5249M_p = 0.32$ pairing	92160 bits
0	51	128	$3650M_p = 0.22$ pairing	156672 bits

Algorithm 5. Partial reduced Tate pairing delegation

```
Input: Elliptic curve $E(\mathbb{F}_p)$ of embedding degree $k$ and prime order $r$ subgroup,
       with degree $d$ twist available, points $P \in E(\mathbb{F}_p)$,
       $Q \in E(\mathbb{F}_{p^k}) \cap \mathrm{Ker}(\pi_{p^{k/d}} - [p^{k/d}])$
Output: Reduced Tate pairing $e_r(P,Q)^{\frac{p^k-1}{r}}$
1 $f = f_{r,P}(Q)$                                    Miller function
2 Compute a random $r$-th power $u \in \mathbb{F}_{p^k}^*$ i.e. such that $\exists v \in \mathbb{F}_{p^k}^*$, $u = v^r$
3 $f' = f \cdot u$
4 Send $f'$ to the external resource
5 Receive $h = (f')^{\frac{p^k-1}{r}} = u^{p^k-1} f^{\frac{p^k-1}{r}} = f^{\frac{p^k-1}{r}} = K$
6 Return $K$
```

Miller function $f_{s,\mathcal{PP}}(\mathcal{SK})$. If Q is actually a public parameter $\mathcal{PP}$, then the line computation $\ell_{\mathcal{PP}}$ can be delegated. The restricted device (such as a smartcard) will ask for the successive intermediate values ℓ then evaluate them at the secret point $P = \mathcal{SK}$.

For an ate pairing on a BN curve, the line is of the form $\ell = \ell_0 + \ell_1\omega + \ell_3\omega^3$, with $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[\omega] = \mathbb{F}_{p^2}[\omega]/(\omega^6 - \xi)$. The smartcard can delegate the computation of the three coefficients then compute the line equation evaluated at $\mathcal{SK}$.

Tangent and line computation. One can found Relic [1] formulas for tangent and line computation in `src/pp/relic_pp_dbl.c` (function `pp_dbl_k12_projc_basic`) and `src/pp/relic_pp_add.c` (function `pp_add_k12_projc_basic`).

We recall the formula from [2, Eq. (10)]:

$$\ell_{2T}(P) = -2YZ\ y_P + 3X^2\ x_P\ \omega + (3b'Z^2 - Y^2)\omega^3 \tag{1}$$

with ω such that $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[\omega]/(\omega^6 - \xi)$, $X, Y, Z \in \mathbb{F}_{p^2}$ and $x_P, y_P \in \mathbb{F}_p$.

The second formula is the following [2, Eq. (13)], with $L = X - x_Q Z$ and $M = Y - y_Q Z$:

$$\ell_{T+Q}(P) = -Ly_P - Mx_P\omega + (MX - LY)\omega^3 \tag{2}$$

In both cases the coefficients of ℓ are computed from a public parameter $Q = \mathcal{PP}$ hence can be delegated. The smart card saves $2S_{p^2} + 7M_{p^2} = 25M_p$. It remains for the smart card to evaluate the line ℓ at $\mathcal{SK} = P = (x_P, y_P)$. This costs $4M_p$ in both cases.

Efficiency improvement. To sum up, the smartcard sends the point $\mathcal{PP}$ to the external computer and computes the intermediate values of the Miller function on the fly, when receiving the coefficients of the intermediate values. No information on $\mathcal{SK}$ is provided to the external helper (except f' which does not reveal more information than the session key $\mathcal{K}$). For an optimal Ate pairing on a Barreto-Naehrig curve, this saves 31 % of the Miller loop, then we delegate 100 % of the final powering, saving at the end 65 % of the pairing cost. Note that the idea can be adapted to achieve (strong) secrecy by further masking the final powering but the efficiency improvement is smaller if pre-computation is not possible. Note also that the same idea can be applied to any instantiation of pairings (but requires a specific analysis).

Acknowledgements. The authors thank Olivier Blazy, Renaud Dubois and Fabien Laguillaumie for their fruitful comments. This work was supported in part by the French ANR-12-INSE-0014 SIMPATIC Project.

References

1. Aranha, D.F., Gouvêa, C.P.L.: RELIC is an Efficient LIbrary for Cryptography, September 2013. http://code.google.com/p/relic-toolkit/
2. Aranha, D.F., Barreto, P.S.L.M., Longa, P., Ricardini, J.E.: The realm of the pairings. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 3–25. Springer, Heidelberg (2014)
3. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
4. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
5. Bos, J.W., Costello, C., Naehrig, M.: Exponentiating in pairing groups. Cryptology ePrint Archive, Report 2013/458 (2013)
6. Boyko, V., Peinado, M., Venkatesan, R.: Speeding up discrete log and factoring based schemes via precomputations. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 221–235. Springer, Heidelberg (1998)
7. Canard, S., Devigne, J., Sanders, O.: Delegating a pairing can be both secure and efficient. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 549–565. Springer, Heidelberg (2014)

8. Chevallier-Mames, B., Coron, J.-S., McCullagh, N., Naccache, D., Scott, M.: Secure delegation of elliptic-curve pairing. Cryptology ePrint Archive, Report 2005/150 (2005)
9. Chevallier-Mames, B., Coron, J.-S., McCullagh, N., Naccache, D., Scott, M.: Secure delegation of elliptic-curve pairing. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 24–35. Springer, Heidelberg (2010)
10. Coron, J.-S., M'Raïhi, D., Tymen, C.: Fast generation of pairs $(k,[k]P)$ for Koblitz elliptic curves. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 151–164. Springer, Heidelberg (2001)
11. Fouque, P.-A., Tibouchi, M.: Indifferentiable hashing to Barreto–Naehrig curves. In: Hevia, A., Neven, G. (eds.) LatinCrypt 2012. LNCS, vol. 7533, pp. 1–17. Springer, Heidelberg (2012)
12. Girault, M., Lefranc, D.: Server-aided verification: theory and practice. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 605–623. Springer, Heidelberg (2005)
13. Granger, R., Scott, M.: Faster squaring in the cyclotomic subgroup of sixth degree extensions. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 209–223. Springer, Heidelberg (2010)
14. Joux, A.: A one round protocol for tripartite Diffie-Hellman. In: Bosma, W. (ed.) ANTS 2000. LNCS, vol. 1838, pp. 385–394. Springer, Heidelberg (2000)
15. Kachisa, E.J., Schaefer, E.F., Scott, M.: Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing 2008. LNCS, vol. 5209, pp. 126–135. Springer, Heidelberg (2008)
16. Nguyen, P.Q., Shparlinski, I.E., Stern, J.: Distribution of modular sums and the security of the server aided exponentiation. In: Lam, K.-Y., Shparlinski, I., Wang, H., Xing, C. (eds.) Cryptography and Computational Number Theory. Progress in Computer Science and Applied Logic, vol. 20, pp. 331–342. Birkhäuser, Basel (2001)
17. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)