IronCore Labs' implementation of the multi-hop PRE algorithm

Jon Lamar and Bob Wall

March 7, 2018

1 The initial algorithm

This document describes parameter choices, algorithms, and details of our implementation of the proxy re-encryption algorithm described in [WC09], and updated in [CL14]. The algorithm and its updates are described in more accessible notation, with examples, in our corresponding document "The original multi-hop proxy re-encryption algorithm".

2 Our choices

Denote the point at infinity of an elliptic curve E by $\mathcal{O}$. We will refer to this point as the *identity* of E (as it is the additive identity under the usual group law on E). All other points of E are called *affine points*.

Our implementation will use the optimal Ate pairing on the Barreto–Naehrig (see [BN06]) elliptic curve E defined by the equation $y^2 = x^3 + 3$ over extensions of the finite field $\mathbb{F}_p$, of order $p = 36t^4 + 36t^3 + 24t^2 + 6t + 1$, where $t = s^3$ and $s = 1868033$. These parameters are chosen to ensure a low Hamming weight of the non-adjacent form of the value $6t + 2$; this value affects the number of adds in the Miller loop used to evaluate the pairing (see [NNS10]). This curve, E,[1] has prime order over the base field $\mathbb{F}_p$, and that prime is $r = 36t^4 + 36t^3 + 18t^2 + 6t + 1$.[2] Thus, all affine points of $E(\mathbb{F}_p)$ have additive order r.

However, the optimal Ate pairing will require $\mathbb{G}_1$ to be the full r-torsion of the elliptic curve $E(\overline{\mathbb{F}_p})$,[3] which is defined to be the set of points of $E(\overline{\mathbb{F}_p})$ with order divisible by r. However, we do not need to resort to infinite extensions: $E(\mathbb{F}_{p^{12}})$ contains the full

[1] We use the notation E for the curve as an algebraic variety and $E(k)$ to talk about the group of k-valued points of E, where k is any extension of $\mathbb{F}_p$.

[2] That p and r are prime is not a consequence of the formulae; t must be chosen carefully.

[3] There is some math jargon here: $\overline{\mathbb{F}_p}$ is the algebraic closure of $\mathbb{F}_p$, which is the union of all of the extensions of $\mathbb{F}_p$. Thus $E(\overline{\mathbb{F}_p})$ will contain $E(\mathbb{F}_{p^k})$ for all k.

r-torsion of $E(\overline{\mathbb{F}_p})$ (see [Cos17, Chapter 4.1]). We use $E(\mathbb{F}_{p^{12}})[r]$ to denote the r-torsion of $E(\mathbb{F}_{p^{12}})$.

Unfortunately, the domain of the optimal Ate pairing e is not all of $\mathbb{G}_1 \times \mathbb{G}_1$, and to describe the restrictions on the input we need to build some terminology. We will require the left entry to be an element of $E(\mathbb{F}_p)[r]$ (i.e., this entry must have entries in $\mathbb{F}_p$), while the right entry must lie in the *trace-zero subgroup* of the r-torsion. These are the points which lie in the kernel of the *trace map*. The trace map Tr is defined as follows: if $P = (x, y)$ is a point of E, then

$$\mathrm{Tr}(P) = \sum_{i=0}^{11} (x^{p^i}, y^{p^i}). \tag{2.1}$$

It is important to note that the addition in the above sum is addition in the elliptic curve, i.e., addition using the chord-and-tangent rule. It is *not* standard vector addition.

Now, note that if q is a power of p, then the finite field $\mathbb{F}_{q^d}$ may be constructed as an extension of $\mathbb{F}_q$ by taking the quotient of the polynomial ring $\mathbb{F}_q[x]$ by the ideal generated by $f(x)$, where $f(x)$ is any irreducible polynomial of degree d over $\mathbb{F}_q$. This object is usually denoted $\mathbb{F}_q[x]/(f(x))$. For our purposes, we will construct $\mathbb{F}_{p^{12}}$ as the top of a tower of extensions of degrees 2 or 3 using polynomials of the form $x^d - c$. More precisely, we will use the following constructions:

$$\begin{aligned} \mathbb{F}_{p^2} &= \mathbb{F}_p[u]/(u^2 + 1) \\ \mathbb{F}_{p^6} &= \mathbb{F}_{p^2}[v]/(v^3 - (u + 3)) \\ \mathbb{F}_{p^{12}} &= \mathbb{F}_{p^6}[w]/(w^2 - v). \end{aligned} \tag{2.2}$$

There is one more alteration that we must make. In order to cut down on computational costs due to finite field arithmetic (which grows as a function of the degree of the extension), we can use the following nice property of E: there exists a related curve, E', for which an efficiently computable group isomorphism $\psi : E'(\mathbb{F}_{p^2}) \to E(\mathbb{F}_{p^{12}})$ exists. For our choice of parameters so far, E' will be the elliptic curve defined by the equation $y^2 = x^3 + 3/(u + 3)$ and the isomorphism ψ is defined by the rule

$$(x, y) \mapsto (x \cdot w^2, y \cdot w^3). \tag{2.3}$$

The algorithms require two *generator points*, one ($\mathbf{g}$) on the curve $E(\mathbb{F}_p)$, and one in the trace zero subgroup ($\mathbf{g}_1$); that is, a point on $E'(\mathbb{F}_{p^2})$. We chose one of the simplest points for $\mathbf{g}$ - $(1, 2)$. Choosing $\mathbf{g}_1$ was more complicated; we followed these steps:

- Pick a point P on E' by choosing $x = 1$ and finding the y value by computing $sqrt(x^3 + 3/(u + 3))$. We arbitrarily chose the positive root.
- Multiply P by the value $r2 = p + p - r$ to get P'; since the order of $E'(\mathbb{F}_{p^2})$ is $r * r2$, $P * r2$ is guaranteed to be a point on the r-torsion.

- Compute the anti-trace of P' to generate an element of the trace-zero subgroup.

3 Implementation

For the remainder of this document, let E be the curve defined above, with all notation as in that section.

3.1 Finite field arithmetic

The key ideas behind our implementation for finite field arithmetic are taken from [BS10] and related papers. We use the sequence of extensions

$$\mathbb{F}_p \subset \mathbb{F}_{p^2} \subset \mathbb{F}_{p^6} \subset \mathbb{F}_{p^{12}}$$

to represent $\mathbb{F}_{p^{12}}$, with the embeddings defined by (2.2). Some of the basic operations were optimized using methods from [DhSD06].

We used Karatsuba multiplication for elements of $\mathbb{F}_{p^2}$ and $\mathbb{F}_{p^{12}}$ (see [KO63]), which is implmented as follows (see [DhSD06]):
Algorithm 3.1 (Karatsuba multiplication in a degree-2 field extension)**.** Input: elements $a = a_0 + a_1 x$ and $b = b_0 + b_1 x$ of the degree-2 extension $k[x]/(x^2 - \beta)$ of the field k. Output: $c = a \cdot b$.

1. Let $t_0 \leftarrow a_0 \cdot a_0$.
2. Let $t_2 \leftarrow a_1 \cdot b_1$.
3. Let $t_1 \leftarrow (a_1 + a_0) \cdot (b_1 + b_0)$.
4. Let $c_0 \leftarrow t_0 + \beta \cdot t_2$.
5. Let $c_1 \leftarrow t_1 - t_2 - t_0$.
6. Return $c = c_0 + c_1 x$.

Inversion in $\mathbb{F}_{p^6}$ is implemented using [MJ17, Algorithm 5.23], which is as follows.
Algorithm 3.2. Input: element $a = a_0 + a_1 v + a_2 v^2$ of $\mathbb{F}_{p^6}$ represented as an element of $\mathbb{F}_{p^2}[v]/(v^3 - (u + 3))$. Output: $c = a^{-1}$.

1. Let $t_0 \leftarrow a_0^2$.
2. Let $t_1 \leftarrow a_1^2$.
3. Let $t_2 \leftarrow a_2^2$.
4. Let $t_3 \leftarrow a_0 \cdot a_1$.
5. Let $t_4 \leftarrow a_0 \cdot a_2$.

6. Let $t_5 \leftarrow a_1 \cdot a_2$.
7. Let $A \leftarrow t_0 - (u+3) \cdot t_5$.
8. Let $B \leftarrow (u+3) \cdot t_2 - t_3$.
9. Let $C \leftarrow t_1 - t_4$.
10. Let $t_6 \leftarrow t_0 \cdot A$.
11. Let $t_6 \leftarrow t_6 + ((u+3) \cdot a_2 \cdot B)$.
12. Let $t_6 \leftarrow t_6 + ((u+3) \cdot a_1 \cdot C)$.
13. Let $F \leftarrow t_6^{-1}$.
14. Let $c_0 \leftarrow A \cdot F$.
15. Let $c_1 \leftarrow B \cdot F$.
16. Let $c_2 \leftarrow C \cdot F$.
17. Return $c = c_0 + c_1 v + c_2 v^2$.

Inversion in $\mathbb{F}_{p^{12}}$ is implemented using [MJ17, Algorithm 5.19], which is as follows.
Algorithm 3.3. Input: element $a = a_0 + a_1 w$ of $\mathbb{F}_{p^{12}}$ represented as an element of $\mathbb{F}_{p^6}[w]/(w^2 - v)$. Output: $c = a^{-1}$.

1. Let $t_0 \leftarrow a_0^2$.
2. Let $t_1 \leftarrow a_1^2$.
3. Let $t_0 \leftarrow t_0 + v \cdot t_1$.
4. Let $t_1 \leftarrow t_0^{-1}$.
5. Let $c_0 \leftarrow a_0 \cdot t_1$.
6. Let $c_1 \leftarrow -a_1 \cdot t_1$.
7. Return $c = c_0 + c_1 v$.

3.2 Pairing

Recall the trace map, Tr, which is defined in (2.1). There is a related map, called the *anti-trace*, defined by the formula

$$\mathrm{aTr}(P) = 12 \cdot P - \mathrm{Tr}(P).$$

The *trace-zero subgroup* of the r-torsion of E is the kernel of the trace map. These are the points P for which $\mathrm{Tr}(P)$ is the identity. We can obtain elements of the trace-zero subgroup by taking the anti-trace of arbitrary elements of the r-torsion.

Let $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$, and $R = (x_R, y_R)$ be nonidentity points (i.e., not equal to the point at infinity) on E represented in affine coordinates. Define

$$\ell_{Q,R}(P) = y_P - y_Q - \lambda \cdot (x_P - x_Q),$$

where λ is defined as follows: if $Q \neq R$, then $\lambda = (y_R - y_Q)/(x_R - x_Q)$ is the slope of the line through Q and R, otherwise if $Q = R$, then $\lambda = 3x_Q^2/2y_Q$ is the slope of the line through Q tangent to E.

For a point $P \in E$ and a positive integer n, we define a function $f_{n,P}$ recursively by the rules $f_{0,P} \equiv 1$ and

$$f_{n,P} = f_{n-1,P} \cdot \frac{\ell_{(n-1)P,P}}{v_{nP}},$$

where $v_Q(R) = x_R - x_Q$ in affine coordinates (see [Cos17, Chapter 3, Chapter 5] for more intuition and properties regarding this function). The *optimal Ate pairing* is the bilinear pairing $e : G \times G \to \mathbb{F}_{p^{12}}$ is defined by the formula

$$e(P,Q) = \left(f_{6t+2,Q}(P) \cdot \ell_{(6t+2)Q,\pi_p(Q)}(P) \cdot \ell_{(6t+2)Q+\pi_p(Q),-\pi_p^2(Q)}(P)\right)^{\frac{p^{12}-1}{r}}, \tag{3.1}$$

where (see Section 2):

1. t is the Barreto–Naehrig parameter $t = (1868033)^3$;
2. $p = 36t^4 + 36t^3 + 24t^2 + 6t + 1$ is the size of the base field $\mathbb{F}_p$;
3. π_p is the Frobenius p-power map: $\pi_p(x, y) = (x^p, y^p)$;
4. $r = 36t^4 + 36t^3 + 18t^2 + 6t + 1$ is the (prime) order of $E(\mathbb{F}_p)$;
5. P is any element of $E(\mathbb{F}_p)$ of order r;
6. Q is any element of the trace-zero subgroup of the r-torsion $E(\mathbb{F}_{p^{12}})[r]$.

Our implementation of the optimal Ate pairing is based on the papers [NNS10], [BGDM+10], [DSD07], and [SBC+09].

3.2.1 Using a sextic twist

The computation of $e(P, Q)$ is broadly split into two steps: the first is the computation of $f_{6t+2,Q}(P)$ and its product with the two line functions of (3.1), and the second is the exponentiation of that product to the power of $(p^{12} - 1)/r$. The computational cost of the first of these two steps can be significantly decreased using the sextic twist ψ as defined in (2.3).

In our implementation, this is done by representing any point of the r-torsion other than P with its preimage under ψ, and precomposing all operations involving these points

with ψ. In particular (and for example) if $Q = (x, y) \in E'(\mathbb{F}_{p^2})[r]$, the Frobenius map π_p becomes

$$\pi'_p(Q) = (\psi^{-1} \circ \pi_p \circ \psi)(Q) = (w^{2(p-1)}x, w^{3(p-1)}y)$$

and the line function $\ell_{Q,R}(P)$ becomes

$$\ell'_{Q,R}(P) = \ell_{\psi(Q),\psi(R)}(P).$$

3.2.2 Miller's algorithm

We directly implemented [BGDM$^+$10, Algorithm 1], which is a non-adjacent form (NAF) version of Miller's algorithm. The algorithm (hereafter referred to as the *Miller loop*) is as follows.

Algorithm 3.4. Input: order r element $P \in E(\mathbb{F}_p)$, trace-zero element $Q \in E'(\mathbb{F}_{p^2})[r]$. Output: optimal Ate pairing $e(P, \psi(Q))$.

1. Write $u = 6t + 2$ in non-adjacent form as $u = \sum_{i=0}^{L-1} u_i 2^i$, where each $u_i \in \{-1, 0, 1\}$.
2. Let $R \leftarrow Q$.
3. Let $f \leftarrow 1$.
4. for $i = L - 2$ down to 0, do:
   (a) Let $(f, R) \leftarrow (f^2 \cdot \ell'_{R,R}(P), 2R)$.
   (b) If $u_i = -1$, then let $(f, R) \leftarrow (f \cdot \ell'_{T,-Q}(P), R - Q)$.
   (c) Else if $u_i = 1$, then let $(f, R) \leftarrow (f \cdot \ell'_{T,Q}(P), R + Q)$.
5. Let $Q_1 \leftarrow \pi'_p(Q)$.
6. Let $Q_2 \leftarrow \pi'_p(Q_1)$.
7. Let $(f, R) \leftarrow (f \cdot \ell'_{R,Q_1}(P), R + Q_1)$.
8. Let $f \leftarrow f \cdot \ell'_{R,-Q_2}(P)$.
9. Let $f \leftarrow f^{\frac{p^4-p^2+1}{r}}$.
10. Return f.

3.2.3 Line functions and projective coordinates

Much of the computational cost of the Miller loop is associated with the evaluation of the line functions $\ell'_{Q,R}(P)$. We save some time here by representing Q and R in homogeneous projective coordinates and P in affine coordinates. This is the method of [DSD07].

3.2.4 Final exponentiation

Once we have computed the inside of (3.1), the remaining step is to raise this value (an element of $\mathbb{F}_{p^{12}}$, which we hereafter denote f) to the power of $(p^{12}-1)/r$. First, we factor $(p^{12}-1)/r$ as $(p^6-1)(p^2+1)(p^4-p^2+1)/r$, so that

$$f^{\frac{p^{12}-1}{r}} = \left((f^{p^6-1})^{p^2+1}\right)^{\frac{p^4-p^2+1}{r}}.$$

The first two exponentiations (in order of operations) of the right-hand side involve exclusively the Frobenius p-power map[4] and inversion on $\mathbb{F}_{p^{12}}$, both of which can be implemented efficiently using standard algorithms. The outermost exponentiation (called the "hard" exponentiation) is implemented be expressing $(p^4-p^2+1)/r$ as a polynomial in the Barreto–Naehrig parameter t and its cube root $s = \sqrt[3]{t} = 1868033$:

$$\frac{p^4-p^2+1}{r} = p^3 + (6t^2+1)p^2 + (-36t^3-18t^2-12t+1)p + (-36t^3-30t^2-18t-2).$$

Thus, the hard exponentiation can be further reduced to the following form:

$$f^{\frac{p^4-p^2+1}{r}} = (f^{p^3}) \cdot (f^{p^2})^{6t^2+1} \cdot (f^p)^{-36t^3-18t^2-12t+1} \cdot f^{-36t^3-30t^2-18t-2}.$$

An efficient algorithm for computing the above expression is as follows.

Algorithm 3.5. [DSD07, Algorithm 3] Input: An element f of $\mathbb{F}_{p^{12}}$. Output: $f^{\frac{p^4-p^2+1}{r}}$.

1. Let $a \leftarrow f^{-6s^3-5}$.
2. Let $b \leftarrow a^p$ using the Frobenius map on $\mathbb{F}_{p^{12}}$.
3. Let $b \leftarrow ab$.
4. Compute f^p, f^{p^2}, and f^{p^3} using Frobenius.
5. Let $f \leftarrow f^{p^3} \cdot [b \cdot (f^p)^2 \cdot f^{p^2}]^{6s^6+1} \cdot b \cdot (f^p \cdot f)^9 \cdot a \cdot f^4$.
6. Return f.

In [GS10, Section 3], the authors note that after the "easy" exponentiation, the variable f and its powers have some nice properties: in particular inversion is the same as complex conjugation and therefore is essentially computationally free. Hence, we can use a non-adjacent form version of the square-and-multiply routine for the powers in the above algorithm (as opposed to a standard square-and-multiply routine). We can also save some time when squaring elements of $\mathbb{F}_{p^{12}}$ using the results of Section 3.1 of that paper.

[4] Maps related to exponentiation by the characteristic of the underlying field are often referred to as "Frobenius" maps for theoretical and historic reasons. For our purposes, this is merely coordinate-wise exponentiation by p.

4 Distributed key generation

We have made an adjustment to the proxy re-encryption algorithm to enhance the revocation capabilities. When we generate keypairs for users or groups, we do a distributed computation, where the user device generates an initial keypair, but the proxy server generates an initial component of the keypair and uses this to augment the user's or group's public key. The proxy server does not disclose the private component of the keypair to the user or any group member. In this way, if someone encrypts a message to the public key, the user or group does not have the necessary private key to actually decrypt the message. It must request a re-encryption of the message to a device that is associated with the user. Device keypairs are not augmented in this way, so the device has a private key that is able to decrypt the message.

In order to allow for distributed key generation, we must alter some of the algorithms of the proxy re-encryption scheme (from the already altered versions described in the other paper on the initial algorithm). The algorithms Enc and ReEnc are unchanged. The KeyGen, ReKeyGen, and Dec algorithms are changed as shown below (additions and modifications are highlighted in red).

NOTE: the changes to the Dec algorithm are only required to allow a private key associated with an augmented public key to decrypt a message. This is only included for completeness; in our implementation, device public keys are never augmented, and we only allow device keys to decrypt messages, so we use the original Dec algorithm.

The algorithm $\mathsf{KeyGen}(params) \rightarrow (pk, sk_{\text{local}}, sk_{\text{proxy}})$: Generate a public/private key pair for a user. On input $params$, the client performs the following steps:

1. Let the *local secret key* sk_{local} be chosen randomly from $\mathbb{Z}_p^*$
2. The *unaugmented public key* is computed by

$$pk' \leftarrow sk_{\text{local}} \cdot \mathbf{g}$$

The public key pk' is transmitted to the proxy P_i that is responsible for re-encrypting user i's messages. P_i then performs the following steps to augment the key:

1. Let the *proxy secret key* sk_{proxy} be chosen randomly from $\mathbb{Z}_p^*$
2. The *public key* is computed by

$$pk \leftarrow pk' + sk_{\text{proxy}} \cdot \mathbf{g}$$

pk is the public key that everyone will use to encrypt messages to the user.[5]

The algorithm $\mathsf{ReKeyGen}(params, sk_{\text{local},i}, \mathbf{pk}_j, psk_i, ssk_i) \rightarrow rk_{i \rightarrow j}$: Generate a re-encryption key from user i (the delegator) to user j (the delegatee), given proxy P_i

[5]The proxy secret key sk_{proxy} is kept by P_i, and the local secret key sk_{local} is kept by the client.

associated to user i. The proxy P_i will perform the actual re-encryption of ciphertexts encrypted to user i after that user has delegated decryption to user j.

1. Let the *re-encryption secret key*, rsk, be chosen randomly from $\mathbb{Z}_p^*$
2. Let the *temporary key* K be chosen randomly from $\mathbb{G}_T$
3. Let the *re-encryption public key* be $\mathbf{rpk} \leftarrow rsk \cdot \mathbf{g}$
4. Let the *encrypted temporary key* be
$$rek \leftarrow K \cdot e(\mathbf{pk}_j, \mathbf{g_1})^{esk}$$
5. Let the *re-encryption point* be
$$\mathbf{rep} \leftarrow \mathbf{H_2}(K) + (-sk_i) \cdot \mathbf{g_1}$$
6. The *signature* is
$$sig \leftarrow \mathcal{S}(\mathbf{rpk}||rek||\mathbf{rep}||psk_i, ssk_i)$$
7. The *re-encryption key* is
$$rk_{i \to j} \leftarrow (\mathbf{rpk}, rek, \mathbf{rep}, psk_i, sig)$$
8. The re-encryption key is sent to the proxy via a secure channel. The proxy then computes
$$\mathbf{rep} \leftarrow \mathbf{rep} + -sk_{\text{proxy},i} \cdot \mathbf{g_1}$$
and updates the re-encryption key with this modified **rep** value before storing the key.

The algorithm $\mathsf{Dec}(params, sk_{\text{local},i}, C^{(l)}, P_i) \rightarrow m$: Decrypt a ciphertext encrypted to user i, using the proxy P_i. This is never needed in our system, because we never attempt to decrypt an encrypted message that has not been transformed, but if we had that use case, we would do the following:

To Decrypt a first-level ciphertext $C^{(1)}$:

1. Parse $C^{(1)}$ into $(\mathbf{epk}, em, ah)$
2. Let
$$m \leftarrow em \cdot e(\mathbf{epk}, (-sk_{\text{local},i}) \cdot \mathbf{g_1})$$
3. The variables **epk** and m are transmitted to the proxy P_i, who computes[6]
$$m \leftarrow m \cdot e(\mathbf{epk}, (-sk_{\text{proxy},i}) \cdot \mathbf{g_1})$$
and returns m as the final plaintext.

[6]We always decrypt on a device, and messages are always encrypted to a group or a user, so we never actually decrypt a first-level ciphertext.

To decrypt a level-l ciphertext $C^{(l)}$ ($l > 1$):

1. Parse $C^{(l)}$ into $(C', RB'^{(2)}, \ldots, RB'^{(l-1)}, RB^{(l)})$
2. Parse C' into $(\mathbf{epk}, em', ah)$
3. Parse $RB^{(l)}$ into $(\mathbf{rpk}^{(l)}, rek^{(l)}, \mathbf{rrpk}^{(l)}, rrek^{(l)}, psk_{proxy}, sig)$
4. For each integer k in $[2, l-1]$, parse $RB'^{(k)}$ into $(\mathbf{rpk}^{(k)}, rek'^{(k)}, \mathbf{rrpk}^{(k)}, rrek'^{(k)}, psk_{proxy}, sig)$
5. Let
$$K_{l-1} \leftarrow rek^{(l)} \cdot e(\mathbf{rpk}^{(l)}, (-sk_{\text{local},i}) \cdot \mathbf{g_1})$$
6. Let
$$rrK_{l-1} \leftarrow rrek^{(l)} \cdot e(\mathbf{rrpk}^{(l)}, (-sk_{\text{local},i}) \cdot \mathbf{g_1})$$
7. The variables K_{l-1}, rrK_{l-1}, $rpk^{(l)}$, and $rrpk^{(l)}$ are transmitted to the proxy P_i, who computes[7]
$$K_{l-1} \leftarrow K_{l-1} \cdot e(\mathbf{rpk}^{(l)}, (-sk_{\text{proxy},i}) \cdot \mathbf{g_1})$$
and
$$rrK_{l-1} \leftarrow K_{l-1} \cdot e(\mathbf{rrpk}^{(l)}, (-sk_{\text{proxy},i}) \cdot \mathbf{g_1})$$
8. For each integer k from $l-2$ down to 1, recover the k-th temporary key K_k via the operation
$$K_k \leftarrow rek^{(k+1)} \cdot e(\mathbf{rpk}^{(k+1)}, -\mathbf{H_2}(K_{k+1}))$$
and recover the k-th randomized re-encryption temporary key rrK_k via the operation
$$rrK_k \leftarrow rrek^{(k+1)} \cdot e(\mathbf{rrpk}^{(k+1)}, -\mathbf{H_2}(rrK_{k+1}) - \mathbf{H_2}(K_{k+1}))$$
9. Finally, recover the plaintext m via the operation
$$m \leftarrow em' \cdot e(\mathbf{epk}, -\mathbf{H_2}(K_1) - \mathbf{H_2}(rrK_1))$$

References

[BGDM+10] Jean-Luc Beuchat, Jorge E. González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. *High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves*, pages 21–39. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[7] Note that this is only necessary if the public key for the last designee was augmented by the proxy. For our cases, the last re-encryption will always be to a device, and the device's public key was not augmented, so this step will not be necessary.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. *Pairing-Friendly Elliptic Curves of Prime Order*, pages 319–331. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[BS10] Naomi Benger and Michael Scott. *Constructing Tower Extensions of Finite Fields for Implementation of Pairing-Based Cryptography*, pages 180–195. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[CL14] Y. Cai and X. Liu. A multi-use CCA-secure proxy re-encryption scheme. *IEEE 12th International Conference on Dependable, Autonomic, and Secure Computing*, 7, 2014.

[Cos17] Craig Costello. Pairings for Beginners, 2017.

[DhSD06] Augusto Jun Devegili, Colm Ó hÉigeartaigh, Michael Scott, and Ricardo Dahab. Multiplication and squaring on pairing-friendly fields, 2006. augusto@ic.unicamp.br 13564 received 13 Dec 2006, last revised 20 Feb 2007.

[DSD07] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. *Implementing Cryptographic Pairings over Barreto-Naehrig Curves*, pages 197–207. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[GS10] Robert Granger and Michael Scott. *Faster Squaring in the Cyclotomic Subgroup of Sixth Degree Extensions*, pages 209–223. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[KO63] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics Doklady*, 7:595, January 1963.

[MJ17] N.E. Mrabet and M. Joye. *Guide to Pairing-Based Cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series. CRC Press, 2017.

[NNS10] Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In *LATINCRYPT 2010*. Springer Verlag, January 2010.

[SBC+09] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. *On the Final Exponentiation for Calculating Pairings on Ordinary Elliptic Curves*, pages 78–88. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[WC09] Hongbing Wang and Zhenfu Cao. A fully secure unidirectional and multi-use proxy re-encryption scheme. *ACM CCS Poster Session*, 2009.