The Case of Small Prime Numbers Versus the Joye-Libert Cryptosystem

George Teşeanu\textsuperscript{1,2} \href{mailto:tgeorge@dcti.ro}{tgeorge@dcti.ro}

\textsuperscript{1} Advanced Technologies Institute  
10 Dinu Vintilă, Bucharest, Romania

\textsuperscript{2} Simion Stoilow Institute of Mathematics of the Romanian Academy  
21 Calea Grivitei, Bucharest, Romania

Abstract. In this paper we study the effect of using small prime numbers within the Joye-Libert public key encryption scheme. We introduce two novel versions and prove their security. We further show how to choose the system’s parameters such that the security results hold. Moreover, we provide a practical comparison between the cryptographic algorithms we introduced and the original Joye-Libert cryptosystem.

Keywords: public key encryption, gap $2^k$-residuosity, provable security

MSC: 11T71, 94A60, 68P25, 11A07, 11A15

1 Introduction

The Joye-Libert cryptosystem was introduced in \cite{7,20} as a generalisation of the Goldwasser-Micali public key encryption scheme \cite{18,19}. The main advantage of the Joye-Libert scheme compared to Goldwasser-Micali is that the first mentioned supports a larger message space, and thus it considerably decreases the expansion of the ciphertext. Regarding security, the authors prove that inverting the encryption function is equivalent to breaking the quadratic residuosity problem modulo $n = pq$, where $p$ and $q$ are prime numbers. Another important security result is that the scheme is semantically secure if the gap $2^k$-residue assumption modulo $n$ holds. We underline that the Joye-Libert cryptosystem is partially homomorphic with respect to message addition and supports ciphertext randomization.

In this paper, we aim at finding a way to improve decryption times for the Joye-Libert scheme. Therefore, we introduce two variants of the system: an unbalanced version and a multiprime one. In the first version we show how to decrease the size of $p$, while keeping the system secure and implicitly decreasing the complexity of decryption. The only cryptosystem related to this version is called the unbalanced RSA \cite{31}. Just like our version, the scope is to reduce $p$, while keeping the system secure.

In the multiprime version, we increase the number of factors while keeping the size of the modulus constant and we manage to prove that inverting encryption
is equivalent to a vectorial form of the quadratic residuosity assumption. We can find only a related cryptosystem in the literature: the multiprime RSA [4, 30]. The philosophy behind the multiprime RSA is the same as ours: uses parallelism to speed up decryption. A bonus of the multiprime Joye-Libert is that we can also use multiple threads to improve encryption time, while in the case of RSA this is not possible.

In the final section of our paper, we analyze the complexity of the two novel variants. Then we compare the decryption time complexities for all versions of the Joye-Libert scheme. If parallelization is possible, then the multiprime variant is to be preferred since it has faster encryption and decryption. Otherwise, the unbalanced version should be used.

Previous work. Note that a preliminary version of this proposal was presented in [27].

Structure of the paper. In Section 2 we provide the notations used in our paper. Then we recall several definitions needed to describe our proposals. The original Joye-Libert scheme is detailed in Section 3. In Sections 4 and 5 we present two novel versions of the Joye-Libert scheme. A performance analysis of the Joye-Libert variants is provided in Section 6. Conclusions and open problems are given in Section 7.

2 Preliminaries

Notations. In this paper, $\lambda$ represents a security parameter. By $|n|$ we denote the size of $n$ in bits. The action of selecting a random element $x$ from a sample space $X$ is denoted by $x \xleftarrow{\$} X$. The assignment operator $x \leftarrow y$ initialises variable $x$ with value $y$. Let $E$ be an event, then $\Pr[E]$ represents the likelihood of $E$ occurring. Probabilistic polynomial-time algorithms are referred to as PPT algorithms. In this paper, the set of natural numbers $\{0, \ldots, a-1\}$ is denoted by $[0, a)$. To simplify notations we denote the set $[0, a+1)$ by $[0, a]$. Multidimensional vectors $v = (v_0, \ldots, v_{s-1})$ are represented as $v = \{v_i\}_{i \in [0, s]}$.

2.1 Computational Complexity

In this subsection we present the reader with computational complexities of multiplication, exponentiation and modular inverse. These complexities are needed in order to determine the efficiency of our proposed schemes. Note that the asymptotic values are given in Table 1 and are taken from [15, 28]. To simplify our presentation, we use the notation $M(\cdot)$ for the complexity of the multiplication algorithm. Note that while discussing the complexity of performing an exponentiation we assume that the exponent’s length is $k$ bits.
### 2.2 Number Theoretic Prerequisites

The Joye-Libert encryption scheme is based on a generalisation of the Legendre symbol, namely the $2^k$-th power residue symbol. Note that the Legendre symbol is obtained when $k = 1$ and, for simplicity, we further denote it by $J_p(a)$. We recall the definition of the $2^k$-th power residue symbol and some of its properties as stated in [33].

**Definition 1.** Let $p$ be an odd prime such that $2^k | p - 1$. Then the symbol

$$J_{p,2^k}(a) = a^{\frac{p-1}{2^k}} \mod p$$

is called the $2^k$-th power residue symbol modulo $p$, where $a^{\frac{p-1}{2^k}} \in \mathbb{Z}_p$, where $\mathbb{Z}_p = \{-\frac{p-1}{2}, \ldots, -1, 0, 1, \ldots, \frac{p-1}{2}\}$.

**Lemma 1.** The $2^k$-th power residue symbol satisfies the following properties

1. If $a \equiv b \mod p$, then $J_{p,2^k}(a) = J_{p,2^k}(b)$;
2. $J_{p,2^k}(a^{2^k}) = 1$;
3. $J_{p,2^k}(ab) = J_{p,2^k}(a)J_{p,2^k}(b) \mod p$;
4. $J_{p,2^k}(1) = 1$ and $J_{p,2^k}(-1) = (-1)^{(p-1)/2^k}$.

Let $n = p_1 \ldots p_t$. We further denote by $J_n(a) = J_{p_1}(a) \ldots J_{p_t}(a)$ the Jacobi symbol of an integer $a$ modulo an integer $n$. $J_n$ and $QR_n$ denote the set of integers modulo $n$ with Jacobi symbol 1 and, respectively, the set of quadratic residues modulo $n$.

### 2.3 Public Key Encryption

The three PPT algorithms specific to a public key encryption (PKE) scheme are: Setup, Encrypt and Decrypt. Given as input a security parameter, the Setup algorithm outputs the public key and the corresponding secret key. To encrypt a message, Encrypt also needs the public key as input in order to output the correlated ciphertext. To recover the original message the Decrypt algorithm takes as input the secret key and the ciphertext. Note that if decryption fails, Decrypt returns an invalidity symbol.

**Definition 2 (Indistinguishability under Chosen Plaintext Attacks - IND-CPA).** The security model against chosen plaintext attacks for a PKE scheme is described using the following game:
Setup(\(\lambda\)): In this phase, challenger \(C\) first computes the public key, while keeping the corresponding secret key to himself. Then \(C\) sends only the public key to adversary \(A\).

Query: Adversary \(A\) chooses two equal length messages \(m_0, m_1\) and sends them to \(C\). After flipping a coin \(b \in \{0, 1\}\), the challenger encrypts \(m_b\) and sends to \(A\) the resulting ciphertext.

Guess: In this phase, the adversary outputs a guess \(b' \in \{0, 1\}\). \(A\) wins the security game if \(b' = b\).

The advantage of an adversary \(A\) attacking a PKE scheme is defined as

\[
ADV_A^{\text{IND-CPA}}(\lambda) = |Pr[b = b'] - 1/2|
\]

where the probability is computed over the random bits used by \(C\) and \(A\). A PKE scheme is IND-CPA secure if for any PPT adversary \(A\), the advantage \(ADV_A^{\text{IND-CPA}}(\lambda)\) is negligible.

3 The Joye-Libert PKE scheme

The Joye-Libert encryption scheme was initially introduced in [20]. Its security was improved in [7]. The authors proved that inverting the encryption function is as hard as the quadratic residuosity assumption. Joye et al. also showed that in the standard model the IND-CPA security of the PKE is equivalent to the gap \(2^k\)-residuosity assumption. We shortly present the algorithms of the Joye-Libert cryptosystem.

Setup(\(\lambda\)): Set an integer \(k \geq 1\). Randomly generate two distinct large prime numbers \(p, q\) such that \(|p| = |q| = \lambda\) and \(p \equiv 1 \mod 2^k\). Output the public key \(pk = (n, y, k)\), where \(n = pq\) and \(y \in J_n \setminus QR_n\). The corresponding secret key is \(sk = (p, q)\).

Encrypt(\(pk, m\)): To encrypt a message \(m \in [0, 2^k)\), we choose \(x \leftarrow Z_p^*\) and compute \(c \equiv y^m x^{2^k} \mod n\). Output the ciphertext \(c\).

Decrypt(\(sk, c\)): Compute the value \(z \equiv J_{p, 2^k}(c)\) and find \(m\) such that the relation \([J_{p, 2^k}(y)]^m = z \mod p\) holds. Efficient methods to recover \(m\) can be found in a subsequent section.

4 The Unbalanced Joye-Libert PKE scheme

In the unbalanced Joye-Libert scheme we reduce the size of \(p\) (denoted by \(\lambda_p\)) while keeping the size of \(n\) constant (denoted by \(\lambda_n\)). This modification only impacts the description of the Setup algorithm, which we briefly describe below. Therefore, we have \(\lambda_n = \lambda_p + \lambda_q\), where \(\lambda_q = |q|\) and \(\lambda_p \leq \lambda_q\). Note that when \(\lambda_p = \lambda_q\) we obtain the Joye-Libert cryptosystem, which we further refer to as the balanced Joye-Libert scheme.
\section{The Multiprime Joye-Libert PKE scheme}

\subsection{Description}

We further describe the multiprime Joye-Libert encryption scheme. In this case we split up \( n \) into multiple primes. Therefore, we have that \( \lambda_n = t \lambda_p + \lambda_q \). Note that when \( \lambda_p = \lambda_q \) and \( t = 1 \) we obtain the original cryptosystem from [7, 20] and, moreover, when in addition we set \( k = 1 \) we obtain the Goldwasser-Micali cryptosystem [18]. Also, if we set \( t = 1 \) we obtain the unbalanced version.

\textit{Setup}(\( \lambda_p, \lambda_q \)): Set an integer \( k \geq 1 \). Randomly generate \( t + 1 \) distinct large prime numbers \( p_1, \ldots, p_t, q \) such that \( |p_1| = \ldots = |p_t| = \lambda_p, |q| = \lambda_q \) and \( p_1, \ldots, p_t \equiv 1 \text{ mod } 2^k \). Let \( n = p_1 \ldots p_t q \). For each \( i \in [1, t] \) select \( y_i \in \mathbb{Z}_n^* \) such that \( J_{p_i}(y_i) = J_q(y_i) = -1 \) and \( J_{p_i, 2^k}(y_i) = 1 \), where \( j \neq i \).

Output the public key \( pk = (n, Y, k) \) and the corresponding secret key \( sk = (P, Y) \), where \( Y = \{y_i\}_{i \in [1, t]} \) and \( P = \{p_i\}_{i \in [1, t]} \).

\textit{Encrypt}(\( pk, m \)): To encrypt message \( m \in [0, 2^k t) \), first we divide it into \( t \) blocks \( m = m_1 \parallel \ldots \parallel m_t \) such that for each \( i \in [1, t] \) we have \( |m_i| = k \). Then, we choose randomly \( x \in \mathbb{Z}_n^* \) and compute the value \( c = x^{k \cdot \prod_{i=1}^t m_i} \text{ mod } n \). The output is ciphertext \( c \).

\textit{Decrypt}(\( sk, c \)): For each \( i \in [1, t] \), compute \( m_i \) using Algorithm 1.

\begin{algorithm}[h]
\caption{Basic decryption algorithm}
\textbf{Input}: The secret prime \( p_i \), the value \( y_i \) and the ciphertext \( c \)
\textbf{Output}: The message block \( m_i \)
1 \( m_i \leftarrow 0 \), \( B \leftarrow 1 \)
2 \textbf{foreach} \( s \in [1, k + 1) \) \textbf{do}
3 \hspace{1em} \( z \leftarrow J_{p_i, 2^s}(c) \)
4 \hspace{1em} \( t \leftarrow J_{p_i, 2^s}(y_i)^{m_i} \text{ mod } p_i \)
5 \hspace{1em} \textbf{if} \( t \neq z \) \textbf{then}
6 \hspace{2em} \( m_i \leftarrow m_i + B \)
7 \hspace{2em} \( B \leftarrow 2B \)
8 \textbf{return} \( m_i \)
\end{algorithm}

Remark 1. Modifying the size of \( p \) does not impact the security proofs discussed in [7, 20]. Therefore, as long as factoring is hard, the unbalanced version is secure. We show how to choose \( \lambda_p \) such that factoring remains difficult in Section 6.
Correctness. Let \( m_i = \sum_{w=0}^{k-1} b_w 2^w \) be the binary expansion of block \( m_i \). Note that
\[
J_{p_i,2^s}(c) = J_{p_i,2^s}(x^{2^t} \cdot \prod_{v=1}^{t} y_v^{m_v}) = J_{p_i,2^s}(y_{i,t}) = J_{p_i,2^s}(y_{i,t}) \sum_{w=0}^{k-1} b_w 2^w
\]
since
1. \( J_{p_i,2^s}(x^{2^s}) = 1 \), where \( 1 \leq s \leq k \);
2. \( J_{p_i,2^s}(y_{j,t}) = 1 \), where \( j \neq i \);
3. \( \sum_{w=0}^{k-1} b_w 2^w = \left( \sum_{w=0}^{s-1} b_w 2^w \right) + 2^s \left( \sum_{w=s}^{k-1} b_w 2^{w-s} \right) \).

As a result, the message block \( m_i \) can be recovered bit by bit using \( p_i \).

5.2 Security Analysis

In this section we first introduce a vectorial generalisation of the quadratic residuosity problem and prove that inverting the encryption function of our proposal is as hard as breaking this assumption. Then we generalise the gap 2^k-residuosity problem stated in [20] and prove the IND-CPA security of our proposal.

Before stating the security assumptions used to prove the security of our proposal, we first define the following sets
\[
Z_n(i) = \{ x \in \mathbb{Z}_n^s \mid J_{p_i,2^s}(x) = 1 \text{ for any } j \neq i \},
QR_n(i) = \{ x \in \mathbb{Z}_n(i) \mid J_{p_i}(x) = 1 \text{ and } J_q(x) = 1 \},
J_n(i) = \{ x \in \mathbb{Z}_n(i) \mid J_{p_i}(x) J_q(x) = 1 \},
\]
where \( n = p_1 \ldots p_t q \) and \( i, j \in [1,t] \).

Definition 3 (Vectorial Quadratic Residuosity - VQR). Choose \( t + 1 \) distinct large prime numbers \( p_1, \ldots, p_t, q \) such that \( |p_1| = \ldots = |p_t| = \lambda_p \), \( |q| = \lambda_q \) and \( p_1, \ldots, p_t \equiv 1 \mod 2^k \). Let \( n = p_1 \ldots p_t q \). Let \( A \) be a PPT algorithm that returns 1 on input \( (x_1, \ldots, x_t, n) \) if \( x_i \in QR_n(i) \). We define the advantage
\[
ADV_{A}^{\lambda_p, \lambda_q}(\lambda_p, \lambda_q) = \left| \Pr[A(x_1, \ldots, x_t, n) = 1 \mid x_i \xleftarrow{\$} QR_n(i) \text{ for } i \in [1,t]\} \right| - \left| \Pr[A(x_1, \ldots, x_t, n) = 1 \mid x_i \xleftarrow{\$} J_n(i) \setminus QR_n(i) \text{ for } i \in [1,t]\} \right|.
\]
The Vectorial Quadratic Residuosity assumption states that for any PPT algorithm \( A \) the advantage \( ADV_{A}^{\lambda_p, \lambda_q}(\lambda_p, \lambda_q) \) is negligible.

Theorem 1. Inverting the encryption function of the multiprime Joye-Libert PKE is intractable if the VQR assumption is intractable.

Proof. Let’s assume that there exists an adversary \( A \) such that given a ciphertext \( c \), it recovers the message \( m \). We construct a PPT algorithms that breaks the VQR assumption. More precisely, on input \((y_1, \ldots, y_t)\) \( B \) computes
c \equiv (y_1 \ldots y_t)^{x_a} \mod n$, where $x \in \mathbb{Z}_n^*$. Remark that if $y_i \in J_n(i) \setminus QR_n(i)$, then $c$ is an encryption of $m_{2^k - 1} = 2^{k-1} \| \ldots \| 2^{k-1}$. If $y_i \in QR_n(i)$, then $y_i \equiv w_t^2 \mod n$, and thus $c \equiv (w_1 \ldots w_t x)^{2^k} \mod n$ is an encryption of $m_0 = 0$. According to the above arguments, on input $(y_1, \ldots, y_t, c)$ algorithm $A$ outputs either $m_{2^k - 1}$ or $m_0$, and thus $B$ outputs $0$ or $1$, respectively. Therefore, $B$ breaks the VQR assumption with non-negligible probability.

**Definition 4 (Vectorial Gap $2^k$-Residuosity - VGR).** Choose $t + 1$ distinct large prime numbers $p_1, \ldots, p_t, q$ such that $|p_1| = \ldots = |p_t| = \lambda_q$, $|q| = \lambda_q$ and $p_1, \ldots, p_t \equiv 1 \mod 2^k$. Let $n = p_1 \ldots p_t q$. Let $A$ be a PPT algorithm that returns $1$ on input $(x_1, \ldots, x_t, k, n)$ if $x_i \in J_n(i) \setminus QR_n(i)$. We define the advantage

$$ADV_A^{VGR} (\lambda_q, \lambda_q) = \left| Pr[A(x_1, \ldots, x_t, k, n) = 1 \mid x_i \in J_n(i) \setminus QR_n(i) \text{ for } i \in [1, t] ] - Pr[A(x_1^{2^k}, \ldots, x_t^{2^k}, k, n) = 1 \mid x_i \in Z_n^* \text{ for } i \in [1, t] ] \right|.$$ 

The Vectorial Gap $2^k$-Residuosity assumption states that for any PPT algorithm $A$, the advantage $ADV_A^{VGR} (\lambda_q, \lambda_q)$ is negligible.

**Theorem 2.** The multiprime Joye-Libert PKE is IND-CPA secure if and only if the VGR assumption is intractable.

**Proof.** The proof of the statement is obtained by simply replacing the distribution of the public key elements $(y_1, \ldots, y_t)$. More precisely, we select randomly the $y_i$ values from the multiplicative subgroup of $2^k$ residues modulo $n$ instead of drawing them from the $J_n(i) \setminus QR_n(i)$ set. Under the VGR assumption, the adversary will not notice this change. Therefore, we removed any link between the ciphertext $c$ and the message $m$, and thus the IND-CPA security follows.

### 5.3 Optimizations

**Setup Optimization.** When choosing the public key we have to meet certain restrictions. An effective way to accomplish this is to first randomly choose the values $y_{i,t+1} \in \mathbb{Z}_n^* \setminus QR_{p_i}$, $w_i \in \mathbb{Z}_q^* \setminus QR_q$ and $w_i^{2^k} \equiv Z_{p_i}^*$. Then compute the elements $y_{i,j} \equiv w_i^{2^k} \mod p_j$. Using the Chinese remainder theorem we compute the desired value $y_i \in \mathbb{Z}_n^*$ such that $y_i \equiv y_{i,t} \mod p_\ell$ for all $\ell \in [1, t]$ and $y_i \equiv y_{i,t+1} \mod q$.

**Decryption Optimization.** In order to speed-up the decryption process, the authors of [21] add an extra restriction when generating the prime factors of $n$, and then use it to simplify decryption. Applying this to the multiprime case, we obtain the fact that we have to generate $p_i$ such that $p_i \not\equiv 1 \mod 2^{k+1}$ holds.
Let $p_i' = (p_i - 1)/2^k$ and $\alpha_i[s] = 2^{k-s}p_i'$. Then the following relation between the ciphertext and plaintext holds

$$c^{\alpha_i[s]} \equiv (x^{2^k} \prod_{v=1}^{t} y_{w,v}^{2^w})^{\alpha_i[s]}$$

$$\equiv y_i^{\alpha_i[s] \sum_{w=1}^{s-1} b_w 2^w}$$

$$\equiv y_i^{b_{s-1} 2^{k-1} \cdot \alpha_i[s] \sum_{w=0}^{s-1} b_w 2^w}$$

$$\equiv (-1)^{b_{s-1}} y_i^{\alpha_i[s](m_i \mod 2^{s-1}) \mod p_i}$$

since

1. $(x^{2^k})^{\alpha_i[s]} = x^{2^k - (p_i - 1)} = 1$;
2. $J_{p_i,2^k}(y_j) = 1$, where $j \neq i$;
3. $\sum_{w=1}^{s-1} b_w 2^w = \left( \sum_{w=0}^{s-1} b_w 2^w \right) + 2^s \left( \sum_{w=s}^{s-1} b_w 2^w \right)$;
4. $J_{p_i}(y_i) = -1$.

Hence, if we precompute $D_i = y_i^{-p_i'}$, then we can recover message block $m_i$. Wrapping it all together we obtain Algorithm 2. Note that when $t = 1$ we obtain [21, Algorithm 3]. The authors also propose three other optimizations [21, Algorithm 4, 5 and 6], but their complexity is similar with Algorithm 3’s complexity.

**Algorithm 2: Optimized decryption algorithm**

**Input:** The secret values $(p_i, D_i)$, the value $y_i$ and the ciphertext $c$

**Output:** The message block $m_i$

1. $m_i \leftarrow 0$, $B \leftarrow 1$, $D \leftarrow D_i$
2. $C \leftarrow J_{p_i,2^k}(c)$
3. foreach $j \in [1, k - 1]$ do
   4. $z \leftarrow C^{2^j} \mod p_i$
   5. if $z \neq 1$ then
      6. $m_i \leftarrow m_i + B$
      7. $C \leftarrow C \cdot D \mod p_i$
      8. $B \leftarrow 2B$, $D \leftarrow D^2 \mod p_i$
   9. if $C \neq 1$ then
      10. $m_i \leftarrow m_i + B$
11. return $m_i$

---

Note that two of these optimizations contain a typo: in line 5, Algorithm 5 and line 6, Algorithm 6 we should have $A^{k-j} \neq C[k-j] \mod p$ instead of $A \neq C[k-j] \mod p$. 
6 Implementation and Performance Analysis

6.1 Parameter Selection

The fastest currently known algorithm for factoring composite numbers is the Number Field Sieve (NFS) [26]. The expected running time of the NFS depends on the size of the modulus \( n \) and not on the size of its factors. More precisely, the expected running time is approximately

\[
L[n] = e^{1.923 \log n^{1/3} (\log \log n)^{2/3}}.
\]

In [25, 26], the authors extrapolate the running time needed to factor a modulus of size \( \lambda_n \) from the computational effort required to factor a 512-bit modulus. Hence, a \( \lambda_n \)-bit modulus offers a security equivalent to a block cipher of \( d \)-bit security if

\[
L[2^{\lambda_n}] \simeq 50 \cdot 2^{d-56} \cdot L[2^{512}]. \tag{1}
\]

Since we start from a secure Joye-Libert PKE and we wish to optimize decryption by decreasing the size of some of the factors of the modulus, while keeping the size of the modulus constant, the NFS cannot be expected to factor \( n \). Unfortunately, this strategy can make the resulting PKEs vulnerable to the Elliptic Curve Method (ECM) [22], if we lower the size of the factors below a certain threshold. Compared to the NFS, the ECM has the running time determined by the size of the smallest factor. Thus, if \( p \) is the smallest factor, then the running time of the ECM is

\[
E[n,p] = (\log_2 n)^2 e^{\sqrt{2 \log p \log \log p}}.
\]

Similarly to the NFS, Lenstra [24] extrapolates the equivalent security provided by a module of size \( \lambda_n \) with the smallest prime of size \( \lambda_p \) to be

\[
E[2^{\lambda_n}, 2^{\lambda_p}] \geq 80 \cdot 2^{d-56} \cdot E[2^{768}, 2^{167}]. \tag{2}
\]

From Equations (1) and (2) we can deduce the following equivalency

\[
E[2^{\lambda_n}, 2^{\lambda_p}] \geq 80 \cdot 2^{\log_2(L[2^{\lambda_n}]/(50 \cdot L[2^{512}]))} \cdot E[2^{768}, 2^{167}]. \tag{3}
\]

A different model for predicting the security against the NFS and the ECM is provided in [11]. Compared to Lenstra’s model, Brent uses known historical factoring records to predict the year a modulus of a given size will be factored. Using the least-squares fit, Brent obtains the following equation for the NFS

\[
D_n^{1/3} = \frac{Y - 1928.6}{13.24} \quad \text{or equivalently} \quad Y = 13.24 \cdot D_n^{1/3} + 1928.6 \tag{4}
\]

and for the ECM

\[
D_p^{1/2} = \frac{Y - 1932.3}{9.3} \quad \text{or equivalently} \quad Y = 9.3 \cdot D_p^{1/2} + 1932.3, \tag{5}
\]
where $D_n$ is the number of digits of the factored modulus and $D_p$ is the number of digits of the largest prime factor found using the ECM.

Using regression analysis, we update Brent’s equations using data points from [3, 5, 8, 9, 12, 13, 16, 17, 23, 29, 32] for the NFS and from [10, 34] for the ECM. These data points are presented in Figures 1 and 2.

Fig. 1: Size of general number factored versus year

Fig. 2: Size of general number factored using ECM versus year
Therefore, the updated equation for the NFS is
\[ D_n^{1/3} = \frac{Y - 1926}{13.97} \] or equivalently \[ Y = 13.97 \cdot D_n^{1/3} + 1926 \] (6)

and for the ECM
\[ D_p^{1/2} = \frac{Y - 1939}{8.207} \] or equivalently \[ Y = 8.207 \cdot D_p^{1/2} + 1939 \] (7)

Equations (4) to (7) are presented in Figures 3 and 4. Note that the black dots represent the acquired data points. We observe that in the case of the NFS the estimates are close, while in the case of the ECM the new estimate is more pessimistic from a security point of view. Using the updated estimates (Equations (6) and (7)) we obtain the following equivalency
\[ D_p^{1/2} = \frac{13.97 \cdot D_n^{1/3} - 13}{8.207} \] (8)

According to NIST [6], the recommended key sizes for composite modules are \( \lambda_n = 1536/3840/15360 \). We preferred to use NIST recommendations instead of the ones from [25,26] since these key sizes are the ones used by the industry and the key sizes from [25,26] are criticized as being too conservative [32]. Therefore, using Equations (3) and (8) we obtain the equivalent size of the smallest prime. The results are presented in Table 2. Note that in the parentheses we provide the maximum number of prime factors that \( n \) can have. Based on these equivalences, we obtain the parameters for the Joye-Libert schemes that offer protection against the NFS and the ECM (see Table 4).
Due to a powerful attack by Coppersmith [14], the size of $k$ must be upper bounded by $0.5\lambda_p$. Otherwise, the factors of $n$ can be found. We can easily see that the block sizes from Table 4 offer a large enough security margin obtained from this bound (see Table 3).

<table>
<thead>
<tr>
<th>Modulus key size</th>
<th>3072</th>
<th>7680</th>
<th>15360</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lenstra model</td>
<td>800(3)</td>
<td>1617(4)</td>
<td>2761(5)</td>
</tr>
<tr>
<td>Regression model</td>
<td>749(4)</td>
<td>1457(5)</td>
<td>2385(6)</td>
</tr>
</tbody>
</table>

Table 2: Equivalent key sizes

| $|p|$ | 1536 | 800 | 749 | 3840 | 1617 | 1457 | 7680 | 2761 | 2385 |
|-----|------|-----|-----|------|------|------|------|------|------|
| $0.5|p|$ | 768 | 400 | 374 | 1920 | 808 | 728 | 3840 | 1380 | 1192 |

Table 3: Coppersmith’s upper bound

6.2 Complexity

Using the complexities provided in Table 1, we compute the asymptotic run times of the decryption algorithm for each Joye-Libert variant. We also determine the size of a block $m_i$ for each variant. The results are provided in Table 5. Note
| $|n|$ | $t$ | $|p|$ | $|q|$ | $|m_i|$ | Type | Model |
|---|---|---|---|---|---|---|
| 3072 | 1 | 1536 | 1536 | 128 | Balanced | - |
| | 1 | 800 | 2272 | 128 | Unbalanced | - |
| | 2 | 800 | 1472 | 64 | Multiprime | - |
| | 1 | 749 | 2323 | 128 | Unbalanced | - |
| | 2 | 749 | 1574 | 64 | Multiprime | - |
| | 3 | 749 | 825 | 43 | - | - |
| 7680 | 1 | 3840 | 3840 | 192 | Balanced | - |
| | 1 | 1617 | 6063 | 192 | Unbalanced | - |
| | 2 | 1617 | 4446 | 96 | Multiprime | - |
| | 3 | 1617 | 2829 | 64 | Multiprime | - |
| | 1 | 1457 | 6223 | 192 | Unbalanced | - |
| | 2 | 1457 | 4766 | 96 | Multiprime | - |
| | 3 | 1457 | 3309 | 64 | Multiprime | - |
| | 4 | 1457 | 1852 | 48 | Multiprime | - |
| 15360 | 1 | 7680 | 7680 | 256 | Balanced | - |
| | 1 | 2761 | 12599 | 256 | Unbalanced | - |
| | 2 | 2761 | 9838 | 128 | Multiprime | - |
| | 3 | 2761 | 7077 | 86 | Multiprime | - |
| | 4 | 2761 | 4316 | 64 | Multiprime | - |
| | 1 | 2385 | 12975 | 256 | Unbalanced | - |
| | 2 | 2385 | 10590 | 128 | Multiprime | - |
| | 3 | 2385 | 8205 | 86 | Multiprime | - |
| | 4 | 2385 | 5820 | 64 | Multiprime | - |
| | 5 | 2385 | 3435 | 52 | Multiprime | - |

Table 4: Joye-Libert parameters' size

| Scheme | $|m|$ | Encryption Complexity | Decryption Complexity |
|---|---|---|---|
| Balanced | $k$ | $\mathcal{O}(2kM(\lambda_n))$ | $\mathcal{O}((2k\lambda + k)M(\lambda))$ |
| | | | $\mathcal{O}((\lambda + k^2/2 + 3k/2)M(\lambda))$ |
| Unbalanced | $k$ | $\mathcal{O}(2kM(\lambda_n))$ | $\mathcal{O}((2k\lambda_p + k)M(\lambda_p))$ |
| | | | $\mathcal{O}((\lambda_p + k^2/2 + 3k/2)M(\lambda_p))$ |
| Multiprime | $tk$ | $\mathcal{O}(2tkM(\lambda_n))$ | $\mathcal{O}((t(2k\lambda_p + k)M(\lambda_p))$ |
| | | | $\mathcal{O}(t(\lambda_p + k^2/2 + 3k/2)M(\lambda_p))$ |
| Parallel Multiprime | $tk$ | $\mathcal{O}(2kM(\lambda_n))$ | $\mathcal{O}((2k\lambda_p + k)M(\lambda_p))$ |
| | | | $\mathcal{O}((\lambda_p + k^2/2 + 3k/2)M(\lambda_p))$ |

Table 5: Performance analysis
that by parallel multiprime we mean the multiprime version in which we use a separate thread to compute each block \(n_i\). We can easily see that for the parameters presented in Table 4, the encryption and decryption complexities of the unbalanced and multiprime versions are similar. Therefore, we only compare the balanced, unbalanced and the parallel multiprime versions. Also, remark that we choose the parameters such that the message spaces are similar for all the variants.

The comparison of the computational complexity of the three variants is presented in Figures 5 to 7. Note that the two sets of crosses for the multiprime version correspond to the two equivalence models: Lenstra - right side crosses and Regression - left side crosses. We also added a dotted red line in the case of the basic decryption (Algorithm 1), which represent the boundary of the optimized decryption algorithm (Algorithm 2) of the balanced version.

From the six plots we can see that the parallel multiprime version always performs better than the (un)balanced version if multiple threads are available. Also, the more threads we use, the faster we recover the original message. Therefore, if additional memory is available and parallelization is possible, then the parallel multiprime version endowed with the optimized decryption algorithm is preferable. Nevertheless, if we only have access to parallelization, then the parallel multiprime version equipped with the basic decryption algorithm is the best choice. Otherwise, we should use the unbalanced variant.

### 6.3 Implementation Details

We further provide the reader with benchmarks for the three Joye-Libert PKE schemes. We ran each of the three sub-algorithms on a CPU Intel i7-4790 4.00 GHz and used GCC to compile it (with the O3 flag activated for optimization). Note that for all computations we used the GMP library [2]. To calculate the running times we used the `omp_get_wtime()` function [1]. For the parallel multiprime variant we used the OMP library [1] to parallelize encryption/decryption. To obtain the average running time in seconds we chose to encrypt 100 128/192/256-bit messages. Therefore, we wanted to simulate a key distribution scenario. The results are provided in Table 6. Note that the optimized version of the decryption algorithm is denoted by `Decrypt (opt)`.

We can see from Table 6 that the conclusions presented in Section 6.2 hold. We can also see that the multiprime version has the shortest time to generate the parameters, while the unbalanced version the longest time. Nevertheless, generating parameters is a one-time operation.

### 7 Conclusions

In this work we introduced two novel versions of the Joye-Libert cryptosystem. The first one, called the unbalanced Joye-Libert PKE, lowers the size of \(p\) in order to decrease decryption time. The second one, called the multiprime Joye-Libert PKE, increases the number of factors and achieves better decryption
Fig. 5: Decryption complexity versus $\lambda_p$ for $\lambda_n = 3072$

Fig. 6: Decryption complexity versus $\lambda_p$ for $\lambda_n = 7680$

Fig. 7: Decryption complexity versus $\lambda_p$ for $\lambda_n = 15360$
times by using multiple threads. Therefore, if parallel threads are available, we recommend the multiprime version, otherwise, we recommend the unbalanced variant. If additional memory is available, then we can replace the basic decryption algorithm with the optimized version, and therefore we can get even better decryption times.

Open Problem. In [7], the authors manage to link the gap $2^k$-residuosity assumption to the quadratic residuosity assumption, when $q \equiv 3 \pmod{4}$ and to the quadratic residuosity and squared Jacobi symbols assumptions, when $q \equiv 1 \pmod{4}$. Therefore, it would be interesting to find a similar link for the VQR assumption. The main bottleneck that we encountered when trying to link it to the VQR is that the probability of choosing an element from $J_n(i)$ is $1/2^{kt-k+2}$, and thus for $t$ elements the probability is $1/2^{kt-k+2}$. Therefore, for practical values of $k$ and $t$, this probability is negligible.

<table>
<thead>
<tr>
<th>$\lvert n \rvert$</th>
<th>$t$</th>
<th>$\lvert p \rvert$</th>
<th>Setup</th>
<th>Encrypt</th>
<th>Decrypt</th>
<th>Decrypt (opt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3072</td>
<td>1</td>
<td>1536</td>
<td>0.193475</td>
<td>0.000782</td>
<td>0.325478</td>
<td>0.007954</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>800</td>
<td>0.414516</td>
<td>0.000783</td>
<td>0.046040</td>
<td>0.002073</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>800</td>
<td>0.129921</td>
<td>0.000584</td>
<td>0.025147</td>
<td>0.001427</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>749</td>
<td>0.507081</td>
<td>0.000782</td>
<td>0.038254</td>
<td>0.001796</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>749</td>
<td>0.140442</td>
<td>0.000567</td>
<td>0.020635</td>
<td>0.001273</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>749</td>
<td>0.036575</td>
<td>0.000419</td>
<td>0.014810</td>
<td>0.000770</td>
</tr>
<tr>
<td>7680</td>
<td>1</td>
<td>3840</td>
<td>6.554910</td>
<td>0.004791</td>
<td>6.516350</td>
<td>0.095550</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1617</td>
<td>14.48600</td>
<td>0.004792</td>
<td>5.65853</td>
<td>0.017291</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1617</td>
<td>6.103730</td>
<td>0.003115</td>
<td>0.287015</td>
<td>0.006300</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1617</td>
<td>1.334260</td>
<td>0.002326</td>
<td>0.205834</td>
<td>0.004104</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1457</td>
<td>17.03100</td>
<td>0.004790</td>
<td>0.390024</td>
<td>0.013370</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1457</td>
<td>6.961240</td>
<td>0.003112</td>
<td>0.201755</td>
<td>0.004917</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1457</td>
<td>2.030210</td>
<td>0.002387</td>
<td>0.162581</td>
<td>0.003348</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1457</td>
<td>0.599723</td>
<td>0.001948</td>
<td>0.154818</td>
<td>0.002867</td>
</tr>
<tr>
<td>15360</td>
<td>1</td>
<td>7680</td>
<td>55.77040</td>
<td>0.018572</td>
<td>47.62750</td>
<td>0.472317</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2761</td>
<td>184.5970</td>
<td>0.018570</td>
<td>3.337000</td>
<td>0.080039</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2761</td>
<td>69.75550</td>
<td>0.010459</td>
<td>1.712070</td>
<td>0.029658</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2761</td>
<td>27.77600</td>
<td>0.007987</td>
<td>1.222490</td>
<td>0.023583</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2761</td>
<td>8.610660</td>
<td>0.006857</td>
<td>0.995357</td>
<td>0.018094</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2395</td>
<td>183.5340</td>
<td>0.018564</td>
<td>2.183190</td>
<td>0.059034</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2395</td>
<td>86.03620</td>
<td>0.010578</td>
<td>1.135120</td>
<td>0.021922</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2395</td>
<td>38.94160</td>
<td>0.008467</td>
<td>0.808199</td>
<td>0.017175</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2395</td>
<td>16.45910</td>
<td>0.008333</td>
<td>0.675422</td>
<td>0.012873</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2395</td>
<td>5.083330</td>
<td>0.006127</td>
<td>0.702819</td>
<td>0.012128</td>
</tr>
</tbody>
</table>

Table 6: Running times
References

1. OpenMP. https://www.openmp.org/
2. The GNU Multiple Precision Arithmetic Library. https://gmplib.org/
34. Zimmermann, P.: 50 Largest Factors Found by ECM. https://members.loria.fr/PZimmermann/records/top50.html