A review of SILBE protocol

https://ia.cr/2024/400

Subham Das

IISER Mohali

June, 2024

Contents

Section 1. Preliminaries and Definitions

Notation

- Let λ be the security parameter.
- We say that $f(x) \leq \text{negl}(x)$ if $|f(x)| \leq x^{-c}$ for any positive integers c for x big enough.
- $\text{PPT}(x)$ is a probabilistic algorithm that is $\text{poly}(x)$.
- Let p be a prime and $\mathbb{F}_p$ is the finite field of charcteristic p.
- Let E and E' be curves over $\overline{\mathbb{F}_p}$

UPKE

Definition

Given λ a security parameter, an UPKE scheme is given by a set of 6 $\mathsf{PPT}(\lambda)$ together with a setup algorithm $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ with pp the public parameters.

$$-\mathsf{KG}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk}) \quad -\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{m}$$

$$-\mathsf{Upk}(\mathsf{pk}, \mu) \to (\mathsf{pk}') \quad -\mathsf{Enc}(\mathsf{pk}, \mathsf{m}) \to \mathsf{ct}$$

$$-\mathsf{Upk}(\mathsf{pp}) \to \mu \quad -\mathsf{Usk}(\mathsf{sk}, \mu) \to \mathsf{sk}'$$

List of Oracles used

1. Upd_list and Cor_list are two lists that respectively store the updates made by the adversaries and what keys are corrupted.
2. Fresh_Upd: It samples a random update μ_i, computes the updated keys $(\mathsf{sk}_{i+1}, \mathsf{pk}_{i+1})$ and return pki+1.
3. Given_Upd: It computes the keys $(\mathsf{sk}_{i+1}, \mathsf{pk}_{i+1})$ corresponding to a given update μ_i and return pk_{i+1}. The update $(i, i+1)$ is added to Upd_list.
4. Corrupt: It receives an index j and return sk_j. It marks j as corrupted together with all others keys of index i such that there is no fresh update in-between.
5. Plaintext_Check the plaintext checking oracle that receives a plaintext and a ciphertext and returns if the ciphertext is a valid encryption of the plaintext.

OW-PCA-U

An UPKE is OW-PCA-U (*One-Wayness under Plaintext CHecking Attack with Updatability*) secure if for any two given adversaries $\mathcal{A}_1, \mathcal{A}_2$ we have that

$$\mathsf{Adv}^{\mathsf{IND-PCA-U}}(\mathcal{A}_1, \mathcal{A}_2) = \mathbb{P}\left[\mathcal{G}^{OW-PCA-U}(\mathcal{A}_1, \mathcal{A}_2) = 1\right] \leq \mathsf{negl}(\lambda)$$

for the following cryptographic game $\mathcal{G}^{OW-PCA-U}$ given by the following figure :

OW-PCA-U

$\mathcal{G}^{\text{OW-PCA-U}}(\mathcal{A}_1, \mathcal{A}_2)$

```
i = 0
Upd_list = Cor_list = ∅
sk_0, pk_0 <-$- KG(1^λ)
j, st <-- A_1^Oracles(pk_0)
if j > i do
    return ⊥
m <-$- M
ct <-$- Enc(pk_j, m)
n <-- A_2^Oracles(ct, st)
if IsFresh(j) do
    return m =? n
return ⊥
```

$\text{Plaintext\_Check}(\text{m}, \text{c}, \text{i}) \to b$

```
if m ∉ M do
    return ⊥
else do
    return m =? Dec(sk_i, c)
```

$\text{IsFresh}(j)$

```
return not j ∈? Cor_list
```

$\text{Fresh\_Upd}() \to \text{pk}_\text{i}$

```
μ <-$- UG(1^λ)
sk_{i+1} <-$- Usk(sk_i, μ)
pk_{i+1} <-$- Upk(pk_i, μ)
i <- i + 1
return pk_i
```

$\text{Given\_Upd}(\mu) \to \text{pk}_\text{i}$

```
sk_{i+1} <-$- Usk(sk_i, μ)
pk_{i+1} <-$- Upk(pk_i, μ)
Upd_list <-- Upd_list ∪ {(i, i+1)}
i <- i + 1
return pk_i
```

$\text{Corrupt}(j) \to \text{sk}_\text{j}$

```
Cor_list = Cor_list ∪ {j}
i, k <- j
while (i − 1, i) ∈ Upd_list do :
    Cor_list = Cor_list ∪ {i − 1}
    i <- i − 1
while (k, k + 1) ∈ Upd_list do :
    Cor_list = Cor_list ∪ {k + 1}
    k <- k + 1
return sk_j
```

Isogeny-Algorithms

- **KernelToIsogeny**
- **Canonical Torsion Basis**
- **PushEndRing**
- **KernelToIdeal**
- **EvalTorsion**
- **RandomEquivalentIdeal**
- **ConstructKani**

M-SIDH scheme

M-SIDH

Alice(pp)		**Bob**(pp)
$s_A \leftarrow_\$ \mathbb{Z}_A, \alpha \leftarrow_\$ \mu_2(B)$		$s_B \leftarrow_\$ \mathbb{Z}_B, \beta \leftarrow_\$ \mu_2(A)$
$R_A \leftarrow P_A + [s_A]Q_A$		$R_B \leftarrow P_B + [s_B]Q_B$
$\phi_A, E_A \leftarrow \textbf{KernelToIsogeny}(E, R_A)$		$\phi_B, E_B \leftarrow \textbf{KernelToIsogeny}(E, R_B)$
$S_A \leftarrow [\alpha]\phi_A(P_B)$		$S_B \leftarrow [\beta]\phi_B(P_A)$
$T_A \leftarrow [\alpha]\phi_A(Q_B)$		$T_B \leftarrow [\beta]\phi_B(Q_A)$
	$\xrightarrow{E_A, S_A, T_A}$	
	$\xleftarrow{E_B, S_B, T_B}$	
$U_A \leftarrow S_B + [s_A]T_B$		$U_B \leftarrow S_A + [s_B]T_A$
$\psi_A, E_K \leftarrow \textbf{KernelToIsogeny}(E_B, U_A)$		$\psi_B, E_K \leftarrow \textbf{KernelToIsogeny}(E_A, U_B)$
$K \leftarrow \mathsf{KDF}(j(E_K))$		$K \leftarrow \mathsf{KDF}(j(E_K))$

Section 2. Constructing a PKE from the generalized lollipop attack

Setup

Algorithm 1 SILBE.Setup

Input: 1^λ

Output: $\mathsf{pp} = (p, (P_0, Q_0), (V_0, U_0), \mathbf{M}_\pi, t)$ with p a prime, $\langle P_0, Q_0 \rangle = E_0[N]$, $\langle U_0, V_0 \rangle = E_0[3^\beta]$, $\mathbf{M}_\pi \in \mathrm{GL}_2(N)$ and t an integer such that $3^{\beta t} \geq p^2$.

Take p a prime of the form $p = 3^\beta N f + 1$ such that $p = 3 \mod 4$ and $N = \prod_{i=1}^{n} p_i$ with p_i distinct odd small prime numbers such that $N \geqslant 3^\beta p^{1/2} \log(p)^2$, N is coprime to 3 and n big enough such that for all $N_k = \prod_{i=k}^{n} p_i$, we have that $N_k \geq \sqrt{3^\beta} \Rightarrow n - k \geq \lambda$.

$P_0, Q_0 \leftarrow \textbf{CanonicalTorsionBasis}(E_0, N)$

$U_0, V_0 \leftarrow \textbf{CanonicalTorsionBasis}(E_0, 3^\beta)$

$\mathbf{M}_\pi \leftarrow \textbf{EvalImageMatrix}(E_0, P_0, Q_0, \pi(P_0), \pi(Q_0))$.

$t \leftarrow \left\lceil \frac{2 \log_2(p)}{\beta \log_2(3)} \right\rceil$

$\mathsf{pp} \leftarrow (p, N, P_0, Q_0, U_0, V_0, \mathbf{M}_\pi, t)$.

return pp

Key Generation

Algorithm 2 **SILBE.KG**

Input: $\mathsf{pp} = (p, (P_0, Q_0), (V_0, U_0), \mathbf{M}_\pi, t)$

Output: pk, sk a public/secret key pair.

1: $J_0 \leftarrow \mathcal{O}_0$
2: **for** $1 \leqslant i \leqslant t$ **do**
3: Sample $\eta_i \in_\$ \mathbb{Z}_{3^\beta}$.
4: $E_i, \rho_i \longleftarrow$ **KernelToIsogeny**$(E_{i-1}, (U_{i-1} + [\eta_i]V_{i-1}))$ ▷ In pp if $i = 1$.
5: $I_i \longleftarrow$ **KernelToIdeal**$(\mathfrak{O}_{E_{i-1}}, (U_{i-1} + [\eta_i]V_{i-1}))$
6: Deterministically compute U_i, V_i a basis of $E_i[3^\beta]$ with $\langle V_i \rangle = \rho_i(E_{i-1}[3^\beta])$.
7: $J_i \longleftarrow$ **RandomEquivalentIdeal**$(J_{i-1}I_i)$
8: **if** $n(J_i) = n(J_{i-1})$ **or** $\widetilde{N}^2 - n(J_i) \neq 1 \mod 4$ **or** is not prime **do**
9: go back to line 7.
10: $S_i, T_i \longleftarrow$ **EvalTorsion**$(\mathfrak{O}_0, \rho_i \circ \kappa_{i-1}, J_{i-1}I_i, id, J_i, \{P_0, Q_0\})$
11: $F_i \longleftarrow$ **ConstructKani**$(n(J_i), \widetilde{N}, \widetilde{N}, (P_0, Q_0, S_i, T_i))$
12: $\mathfrak{O}_{E_i} \longleftarrow$ **PushEndRing**$(\mathfrak{O}_0, \kappa_i, J_i)$ ▷ $\kappa_i \leftarrow F_i(0, 0, -, 0)_3$
13: $I_{\phi_A} \longleftarrow$ **RandomEquivalentIdeal**(J_t)
14: **if** $N' - n(I_{\phi_A})^2 3^{2\beta} \neq 1 \mod 4$ **or** is not prime **do** go back to line 13.
15: $K, L \longleftarrow$ **EvalTorsion**$(\mathcal{O}_0, \rho_t \circ \cdots \circ \rho_1, I_1 \cdots I_t, 1, I_{\phi_A}, P_0, Q_0)$
16: $\mathbf{M}_{\phi_\mathbf{A}} \longleftarrow$ **EvalImageMatrix**(E_t, N, P_t, Q_t, K, L)
17: $\mathsf{pk} \longleftarrow (E_t = E_A)$
18: $\mathsf{sk} \longleftarrow (\mathfrak{O}_{E_t}, I_{\phi_A}, \mathbf{M}_{\phi_A})$
19: **return** pk, sk.

Key Generation

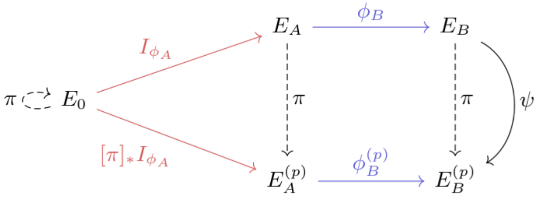
Theorem

Let $\phi : E \to E'$ be an ℓ^h-isogeny obtained from a non backtracking[1] random ℓ-isogeny walk over $\mathcal{G}_p^\ell$. Then for all $\epsilon \in]0,2]$, the distribution of E' has statistical distance $O(p^{-\frac{\epsilon}{2}})$ to the uniform distribution in the supersingular isogeny graph, provided that $h \geq (1+\epsilon)\log_\ell(p)^2$

[1] Given an arbitrary assignment on $\mathcal{G}_\ell$, two consecutive edges φ_0 and φ_1 are said to be backtracking if $\varphi_1 \circ \varphi_0 = [\ell]$, up to possible post-composition by an automorphism. A closed walk is considered to contain backtracking if any consecutive edges, including the last and first, are backtracking.

[2] Proof is referred to `https://eprint.iacr.org/2023/436`

Encryption and Decryption



Encryption

Algorithm 3 SILBE.Enc

Input: $\mathsf{pp}, \mathsf{pk}, \mathsf{m} = (p, (P_0, Q_0), (V_0, U_0), \mathbf{M}_\pi, t), E_A$ with $\mathsf{m} \in \mu_2(N)$

Output: $\mathsf{ct} = (E_B, R_1, R_2)$ with $R_1, R_2 \in E_B[N]$.

$P_A, Q_A \longleftarrow$ **CanonicalTorsionBasis**(E_A, N)

$U_A, V_A \longleftarrow$ **CanonicalTorsionBasis**$(E_A, 3^\beta)$

Sample $r_B \in_\$ \mathbb{Z}_{3^\beta}$

$E_B, \phi_B \longleftarrow$ **KernelToIsogeny**$(E_A, (U_A + [r_B]V_A))$

$\binom{R_1}{R_2} \longleftarrow [\mathsf{m}]\phi_B\binom{P_A}{Q_A}$

$\mathsf{ct} \longleftarrow (E_B, R_1, R_2)$

return ct

Decryption

Algorithm 4 SILBE.Dec

Input: pp, sk, ct $= \big(p, (P_0, Q_0), (V_0, U_0), \mathbf{M}_\pi, t\big), (\mathfrak{O}_{E_A}, I_{\phi_A}, \mathbf{M}_{\phi_\mathbf{A}}), (E_B, \mathsf{R}_1, \mathsf{R}_2)$

Output: m

$P_A, Q_A \longleftarrow$ **CanonicalTorsionBasis**(E_A, N)

$U_B, V_B \longleftarrow$ **CanonicalTorsionBasis**$(E_B^{(p)}, 3^\beta)$

$\binom{S}{T} \longleftarrow \mathbf{M}_{\phi_\mathbf{A}} \binom{R_1}{R_2}$

$\binom{K}{L} \longleftarrow [n(I_{\phi_A})3^\beta]\mathbf{M}_\pi^{-1} \pi \binom{S}{T}$

$G, H \longleftarrow$ **EvalKani**$\big(n(I_{\phi_A})^2 3^{2\beta}, N, N/p_1, S, T, K, L, U_B, V_B\big)$ $\triangleright\ \widehat{\psi} = F(-, 0, 0, 0)_1$

$\widehat{\phi_B} \longleftarrow$ **KernelToIsogeny**$(E_B, G + H)$ $\triangleright$ if $G = H$, take just G

return $(3^\beta)^{-1} \cdot \big($**discretelog**$(P_A, \widehat{\phi_B}(R_1), N)\big) \mod N$

Section 3. PKE Security

PKE Security

SILBE is not IND-CPA (Indistinguishability under chosen-plaintext attack), since distinguishing messages m_0 and m_1, we have to multiply R_1 and R_2 by m_0 and use EvalKani in dimension 8. If one is able to retrieve ϕ_B then this means that the encrypted message was m_0, as that would induce that

$$[m_0]R_1 = [m_0^2]\phi_B(P) = \phi_B(P)$$

The above implies that the adversary can simulate the oracle Plaintext_Check.

PKE Security

Theorem

The security of SILBE as an OW-PCA PKE reduces to the M-SIDH problem over random curves.

Proof. Simulating the Plaintext_Check oracle we have that SILBE is OW-PCA secure $\iff$ SILBE is OW-CPA secure. Following Theorem 1, we have that the distribution of the public key E_A is computationally indistinguishable from the uniform distribution of the supersingular curves since the former is $O(p^{-1/2})$ close to the latter. Let $\mathcal{A}^{\mathrm{OW-CPA}}$ be an adversary. Then construct the following algorithm $\mathcal{B}$ which solves the M-SIDH problem over random curves

1. $\mathcal{B}$ receives (P, Q, S, T) with P, Q the canonical basis of $E[N]$ and $\begin{pmatrix} S \\ T \end{pmatrix} = [m]\varphi \begin{pmatrix} P \\ Q \end{pmatrix}$ with $\varphi : E \to E'$ an isogeny of degree 3^β.
2. $\mathcal{B}$ calls $\mathcal{A}^{\mathrm{OW-CPA}}(E, E', S, T)$ and recieve $n \in \mu_2(N)$
3. It then computes $[n]S, [n]T$ and use EvalKani in dim 8 over these points, retrieving $\ker(\varphi)$. Since 3^β is smooth, using KernelToIsogeny, it can compute φ

The algorithm entails that if $\mathcal{A}^{\mathsf{OW-CPA}}$ succeeds then so does $\mathcal{B}$ that is

$$\mathbb{P}[\mathcal{B} \text{ solves problem 1}] \geq \mathsf{Adv}^{\mathsf{OW-CPA}}(\mathcal{A}^{\mathsf{OW-CPA}})$$

Hence, under the assumption that the M-SIDH problem is hard, SILBE is OW-PCA secure. □

Section 4. Extending PKE to UPKE

SILBE PKE $\to$ UPKE

The authors claim that SILBE key update mechanism relies on the following two properties:

- It can be adapted to any curve E, provided we know an isogeny $\phi : E_0 \to E$
- That one can find the public key by using KernelToIsogeny, without knowledge of $\phi : E_0 \to E$

SILBE PKE $\to$ UPKE

The authors claim that SILBE key update mechanism relies on the following two properties:

- It can be adapted to any curve E, provided we know an isogeny $\phi : E_0 \to E$
- That one can find the public key by using KernelToIsogeny, without knowledge of $\phi : E_0 \to E$

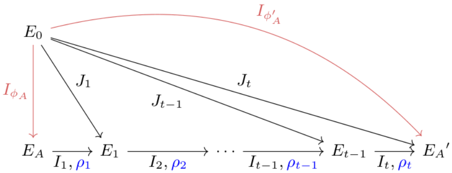


Fig. 5. Diagram of the key update mechanism of SILBE, Alice in red and Bob in blue. Black isogenies are used for the construction of **SILBE.Usk**.

UG: Generates a seed $\mu \in \{0,1\}^{4\log(p)}$

UG: Generates a seed $\mu \in \{0,1\}^{4\log(p)}$

Algorithm 5 SILBE.UG

Input: $\mathsf{pp} = (p, (P_0, Q_0), (V_0, U_0), \mathbf{M}_\pi, t)$
Output: μ an update.
1: Sample $\mu \in_\$ \{0,1\}^{4\log(p)}$ ▷ $4\log(p)$ ensures that H resists quantum attacks.
2: **return** μ

Upk: Use a hash function over μ to generate a sequence of elements in $\mathbb{Z}_{3^\beta}$. Use this sequence to create kernels of an isogeny walk starting at the public key E_A. Using KernelToIsogeny, compute the end curve of that walk, defined as E'_A, the updated public key.

Upk: Use a hash function over μ to generate a sequence of elements in $\mathbb{Z}_{3\beta}$. Use this sequence to create kernels of an isogeny walk starting at the public key E_A. Using KernelToIsogeny, compute the end curve of that walk, defined as E'_A, the updated public key.

Algorithm 6 SILBE.Upk

Input: $\mathsf{pp}, \mathsf{pk}, \mu = \big(p, (P_0, Q_0), (V_0, U_0), \mathbf{M}_\pi, t\big), E_A$.
Output: pk' the updated public key.

$E_0 \leftarrow E_A \qquad U_0, V_0 \longleftarrow$ **CanonicalTorsionBasis**$(E_A, 3^\beta)$
$(\eta_1, \cdots, \eta_t) \leftarrow H(\mu)$ $\quad \triangleright\ \eta_i \in \mathbb{Z}_{3^\beta}$
for $1 \leqslant i \leqslant t$ **do**
  $E_i, \rho_i \longleftarrow$ **KernelToIsogeny**$\big(E_{i-1}, (U_{i-1} + [\eta_i]V_{i-1})\big)$
  Deterministically compute U_i, V_i a basis of $E_i[3^\beta]$ with $\langle V_i \rangle = \rho_i(E_{i-1}[3^\beta])$.
$\mathsf{pk}' \leftarrow E_t = E'_A$
return pk'

Usk: Use a hash function over μ to generate a sequence of elements in $\mathbb{Z}_{3^\beta}$. Use this sequence to create kernels of an isogeny walk starting at the public key E_A. Using KernelToIsogeny, we compute the end curve of that walk defined as E'_A. Using $\phi_A : E_0 \to E_A$, we construct, using EvalKani and RandomEquivalentIdeal, an isogeny $\phi'_A : E_0 \to E_A$, the updated secret key.

Algorithm 7 SILBE.Usk

Input: pp, sk , $\mu = (p, (P_0, Q_0), (V_0, U_0), \mathbf{M}_\pi, t), (\mathfrak{O}_{E_A}, I_{\phi_A}, \mathbf{M}_{\phi_\mathbf{A}}), \mu$
Output: sk$'$ the updated secret key.

1: $E_0 \leftarrow E_A \quad J_0 \leftarrow I_\phi \quad U_0, V_0 \leftarrow$ **CanonicalTorsionBasis**$(E_A, 3^\beta)$
2: $(\eta_1, \cdots, \eta_t) \leftarrow H(\mu)$ ▷ $\eta_i \in \mathbb{Z}_{3^\beta}$
3: **for** $1 \leqslant i \leqslant t$ **do**
4: $\quad E_i, \rho_i \longleftarrow$ **KernelToIsogeny**$(E_{i-1}, (U_{i-1} + [\eta_i]V_{i-1}))$
5: $\quad I_i \longleftarrow$ **KernelToIdeal**$(\mathfrak{O}_{E_{i-1}}, (U_i + [\eta_i]V_i))$
6: $\quad$ Deterministically compute U_i, V_i a basis of $E_i[3^\beta]$ with $\langle V_i \rangle = \rho_i(E_{i-1}[3^\beta])$.
7: $\quad J_i \longleftarrow$ **RandomEquivalentIdeal**$(J_{i-1}I_i)$
8: $\quad$ **if** $n(J_i) = n(J_{i-1})$ **or** $\widetilde{N}^2 - n(J_i) \neq 1 \mod 4$ **or** is not prime **do**
9: $\quad\quad$ go back to line 7.
10: $\quad S_i, T_i \longleftarrow$ **EvalTorsion**$(\mathfrak{O}_0, \rho_i \circ \kappa_{i-1}, J_{i-1}I_i, 1, J_i, P_0, Q_0)$ ▷ Use $\mathbf{M}_\phi$ if $i = 1$
11: $\quad F_i \longleftarrow$ **ConstructKani**$(n(J_i), \widetilde{N}, \widetilde{N}, P_0, Q_0, S_i, T_i)$
12: $\quad \mathfrak{O}_{E_i} \longleftarrow$ **PushEndRing**$(\mathfrak{O}_0, \kappa_i, J_i)$ ▷ $\kappa_i = F(0, 0, -, 0)_3$
13: $I_{\phi'_A} \longleftarrow$ **RandomEquivalentIdeal**(J_t)
14: **if** $N' - n(I_{\phi'_A})^2 3^{2\beta} \neq 1 \mod 4$ **or** is not prime **do** go back to line 12.
15: $K, L \longleftarrow$ **EvalTorsion**$(\mathfrak{O}_0, \kappa_t, J_t, 1, I_{\phi'}, P_0, Q_0)$
16: $\mathbf{M}_{\phi'_\mathbf{A}} \longleftarrow$ **EvalImageMatrix**(E_t, N, P_t, Q_t, K, L)
17: sk$' \longleftarrow (\mathfrak{O}_{E_t}, I_{\phi'_A}, \mathbf{M}_{\phi'_\mathbf{A}})$
18: **return** sk$'$.

Section 5. UPKE Security

Security of SILBE as OW-PCA-U

The authors claim that SILBE remains secure as an UPKE due to the fact that in the Random Oracle Model (ROM), SILBE.Upk is an one way mechanism such that the distribution of the updated public key E'_A is statistically close from the uniform distribution and thus from the public key distribution E_A given by SILBE.KG. Thus, breaking SILBE in the OW-PCA-U scenario would also imply that a fresh instance of SILBE in OW-PCA scenario is also breakable. This leads to following theorem.

Security of SILBE as OW-PCA-U

The authors claim that SILBE remains secure as an UPKE due to the fact that in the Random Oracle Model (ROM), SILBE.Upk is an one way mechanism such that the distribution of the updated public key E'_A is statistically close from the uniform distribution and thus from the public key distribution E_A given by SILBE.KG. Thus, breaking SILBE in the OW-PCA-U scenario would also imply that a fresh instance of SILBE in OW-PCA scenario is also breakable. This leads to following theorem.

Theorem

In the ROM, SILBE is OW-PCA secure $\iff$ SILBE is OW-PCA-U secure

Section 6. Parameters and Efficiency

The authors state that to find "SILBE friendly" primes one would need to find suitable N and β as follows:

- If $N \leq 3^{\beta}\sqrt{p}\log(p) \equiv 3^{3\beta/2}N^{1/2}(\log N + \beta \log 3)$, we increase the size of N.
- If $N_t \geq 3^{\beta/2}$ and $n - t < \lambda$ then we increase the size of β
- After choosing such N and β, we find a good co-factor f such that $p = 3^{\beta}Nf + 1$ is prime.

The authors state that to find "SILBE friendly" primes one would need to find suitable N and β as follows:

- If $N \leq 3^{\beta}\sqrt{p}\log(p) \equiv 3^{3\beta/2}N^{1/2}(\log N + \beta\log 3)$, we increase the size of N.
- If $N_t \geq 3^{\beta/2}$ and $n - t < \lambda$ then we increase the size of β
- After choosing such N and β, we find a good co-factor f such that $p = 3^{\beta}Nf + 1$ is prime.

The primary drawback as stated by the authors is **Efficiency**, which is a result of size of the parameters, together with performing Kani in dim 4 with relatively large primes.

The authors state that to find "SILBE friendly" primes one would need to find suitable N and β as follows:

- If $N \leq 3^{\beta}\sqrt{p}\log(p) \equiv 3^{3\beta/2}N^{1/2}(\log N + \beta\log 3)$, we increase the size of N.
- If $N_t \geq 3^{\beta/2}$ and $n - t < \lambda$ then we increase the size of β
- After choosing such N and β, we find a good co-factor f such that $p = 3^{\beta}Nf + 1$ is prime.

The primary drawback as stated by the authors is **Efficiency**, which is a result of size of the parameters, together with performing Kani in dim 4 with relatively large primes.
For example, number of field operations needed to perform the EvalKani in SILBE.Dec is in the order of $7^5\lambda^5\log(\lambda)^4$, which is, for $\lambda = 128$, around 2^{60}.