

Loop-abort faults on supersingular isogeny cryptosystems

Alexandre Gélín Benjamin Wesolowski

Laboratoire d'Informatique de Paris 6 – Sorbonne Universités UPMC, France

École Polytechnique Fédérale de Lausanne, EPFL IC LACAL, Switzerland

PQCrypto 2017 – Utrecht

2017/06/26

Supersingular-Isogeny Public-key Cryptography

- Introduced by Jao, De Feo, and Plût in 2011

Supersingular-Isogeny Public-key Cryptography

- Introduced by Jao, De Feo, and Plût in 2011
- Based on the same problem as the hash function of [CLG06]

Supersingular-Isogeny Public-key Cryptography

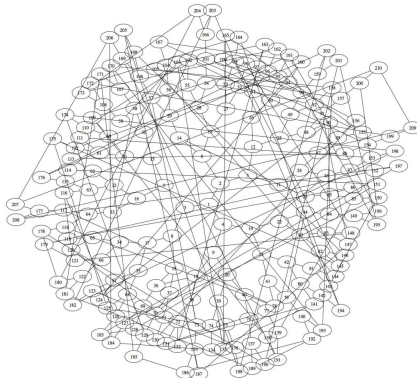
- Introduced by Jao, De Feo, and Plût in 2011
- Based on the same problem as the hash function of [CLG06]

The **isogeny graph** of a supersingular elliptic curve:

Supersingular-Isogeny Public-key Cryptography

- Introduced by Jao, De Feo, and Plût in 2011
- Based on the same problem as the hash function of [CLG06]

The **isogeny graph** of a supersingular elliptic curve:



Definition

A **supersingular** elliptic curve is a curve E defined over \mathbf{F}_{p^k} such that

$$\#E(\mathbf{F}_{p^k}) = 1 \pmod{p}.$$

Definition

A **supersingular** elliptic curve is a curve E defined over \mathbf{F}_{p^k} such that

$$\#E(\mathbf{F}_{p^k}) = 1 \pmod{p}.$$

Interesting properties:

- All supersingular elliptic curves can be defined over \mathbf{F}_{p^2}
- About $\frac{p}{12}$ supersingular elliptic curves, up to isomorphism

Definition

An **isogeny** ϕ between two elliptic curves E_1 and E_2 is a surjective group homomorphism with a finite kernel. The **degree** is defined by

$$\deg \phi = \#\text{Ker } \phi.$$

Definition

An **isogeny** ϕ between two elliptic curves E_1 and E_2 is a surjective group homomorphism with a finite kernel. The **degree** is defined by

$$\deg \phi = \#\text{Ker } \phi.$$

Interesting properties:

- $G \subset E_1 \implies$ a unique E_2 and ϕ such that

$$\phi : E_1 \rightarrow E_2 \quad \text{and} \quad \text{Ker } \phi = G$$

- $E_2 = E/G$ is obtained in $O(\deg \phi)$

Key-Exchange Protocol

- A prime p such that $p+1 = \ell_A^n \ell_B^m$

Key-Exchange Protocol

- A prime p such that $p+1 = \ell_A^n \ell_B^m$
- A supersingular elliptic curve E with $\ell_A^n \ell_B^m$ points

E

Key-Exchange Protocol

- A prime p such that $p+1 = \ell_A^n \ell_B^m$
- A supersingular elliptic curve E with $\ell_A^n \ell_B^m$ points
- A point R_A chosen randomly in $E[\ell_A^n]$

E

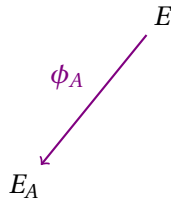
Key-Exchange Protocol

- A prime p such that $p+1 = \ell_A^n \ell_B^m$
- A supersingular elliptic curve E with $\ell_A^n \ell_B^m$ points
- A point R_A chosen randomly in $E[\ell_A^n]$
→ $(m_A, n_A) \in \{1, \dots, \ell_A^n\}^2$ random,
 $R_A = m_A P_A + n_A Q_A$ for $\langle P_A, Q_A \rangle = E[\ell_A^n]$

E

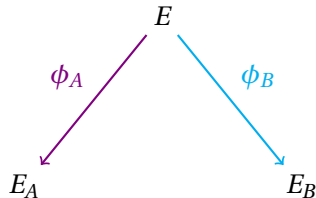
Key-Exchange Protocol

- A prime p such that $p+1 = \ell_A^n \ell_B^m$
- A supersingular elliptic curve E with $\ell_A^n \ell_B^m$ points
- A point R_A chosen randomly in $E[\ell_A^n]$
 - $(m_A, n_A) \in \{1, \dots, \ell_A^n\}^2$ random,
 - $R_A = m_A P_A + n_A Q_A$ for $\langle P_A, Q_A \rangle = E[\ell_A^n]$
 - ⇒ the curve $E_A = E/\langle R_A \rangle$ and $\phi_A: E \rightarrow E_A$



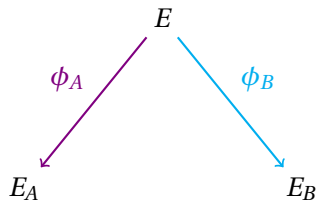
Key-Exchange Protocol

- A prime p such that $p+1 = \ell_A^n \ell_B^m$
- A supersingular elliptic curve E with $\ell_A^n \ell_B^m$ points
- A point R_A chosen randomly in $E[\ell_A^n]$
 - $(m_A, n_A) \in \{1, \dots, \ell_A^n\}^2$ random,
 - $R_A = m_A P_A + n_A Q_A$ for $\langle P_A, Q_A \rangle = E[\ell_A^n]$
 - ⇒ the curve $E_A = E/\langle R_A \rangle$ and $\phi_A: E \rightarrow E_A$
- A point $R_B = m_B P_B + n_B Q_B$ random in $E[\ell_B^m] = \langle P_B, Q_B \rangle$,
the curve $E_B = E/\langle R_B \rangle$ and $\phi_B: E \rightarrow E_B$



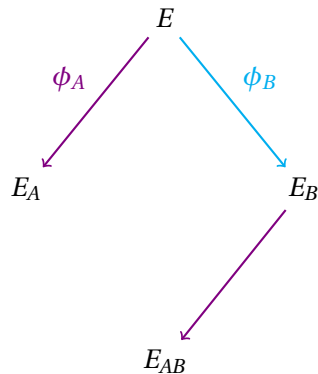
Key-Exchange Protocol

- Bob sends $(E_B, \phi_B(P_A), \phi_B(Q_A))$
where $\langle \phi_B(P_A), \phi_B(Q_A) \rangle = E_B[\ell_A^n]$



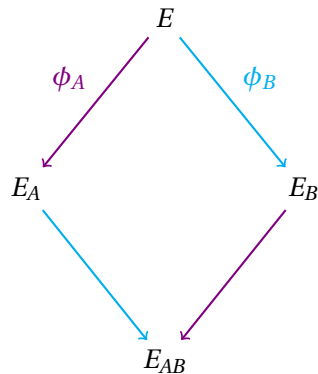
Key-Exchange Protocol

- Bob sends $(E_B, \phi_B(P_A), \phi_B(Q_A))$
where $\langle \phi_B(P_A), \phi_B(Q_A) \rangle = E_B[\ell_A^n]$
- Alice computes $E_{AB} = E_B / \langle m_A \phi_B(P_A) + n_A \phi_B(Q_A) \rangle$



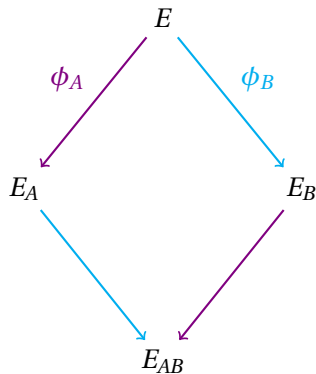
Key-Exchange Protocol

- Bob sends $(E_B, \phi_B(P_A), \phi_B(Q_A))$
where $\langle \phi_B(P_A), \phi_B(Q_A) \rangle = E_B[\ell_A^n]$
- Alice computes $E_{AB} = E_B / \langle m_A \phi_B(P_A) + n_A \phi_B(Q_A) \rangle$
- Bob computes $E_{BA} = E_A / \langle m_B \phi_A(P_B) + n_B \phi_A(Q_B) \rangle$



Key-Exchange Protocol

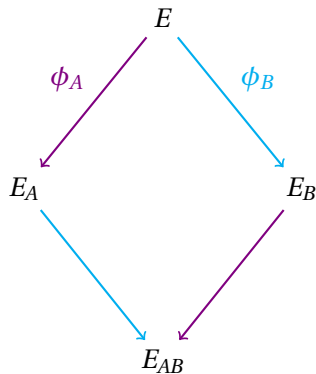
- Bob sends $(E_B, \phi_B(P_A), \phi_B(Q_A))$
where $\langle \phi_B(P_A), \phi_B(Q_A) \rangle = E_B[\ell_A^n]$
- Alice computes $E_{AB} = E_B / \langle m_A \phi_B(P_A) + n_A \phi_B(Q_A) \rangle$
- Bob computes $E_{BA} = E_A / \langle m_B \phi_A(P_B) + n_B \phi_A(Q_B) \rangle$
- $E_{AB} \simeq E / \langle R_A, R_B \rangle \simeq E_{BA}$ so $j(E_{AB}) = j(E_{BA})$



Key-Exchange Protocol

- Bob sends $(E_B, \phi_B(P_A), \phi_B(Q_A))$
where $\langle \phi_B(P_A), \phi_B(Q_A) \rangle = E_B[\ell_A^n]$
- Alice computes $E_{AB} = E_B / \langle m_A \phi_B(P_A) + n_A \phi_B(Q_A) \rangle$
- Bob computes $E_{BA} = E_A / \langle m_B \phi_A(P_B) + n_B \phi_A(Q_B) \rangle$
- $E_{AB} \simeq E / \langle R_A, R_B \rangle \simeq E_{BA}$ so $j(E_{AB}) = j(E_{BA})$

 $\implies j(E_{AB})$ secret shared by Alice and Bob



Position of the problem

Path-finding problem

Given two isogenous curves E_1 and E_2 , find an isogeny between them of degree ℓ_A^n .

Path-finding problem

Given two isogenous curves E_1 and E_2 , find an isogeny between them of degree ℓ_A^n .

- Equivalent to find a path of fixed length in the isogeny graph

Path-finding problem

Given two isogenous curves E_1 and E_2 , find an isogeny between them of degree ℓ_A^n .

- Equivalent to find a path of fixed length in the isogeny graph
- Brute-force attack in $O(\ell_A^n) \approx O(\sqrt{p})$

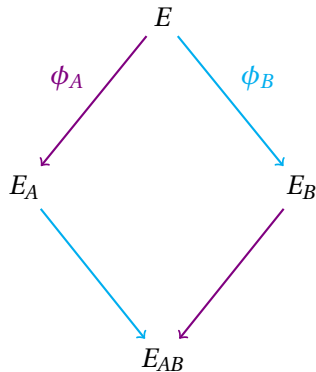
Path-finding problem

Given two isogenous curves E_1 and E_2 , find an isogeny between them of degree ℓ_A^n .

- Equivalent to find a path of fixed length in the isogeny graph
- Brute-force attack in $O(\ell_A^n) \approx O(\sqrt{p})$
- **Claw finding:** Find a collision in $O(\ell_A^{\frac{n}{2}}) \approx O(\sqrt[4]{p})$

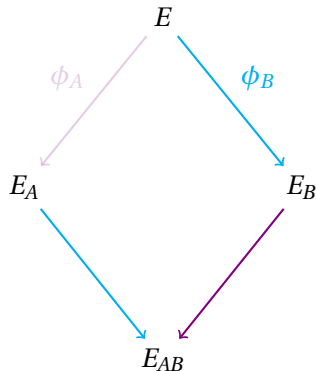
Attack framework

- Alice uses a **static** private key (m_A, n_A)



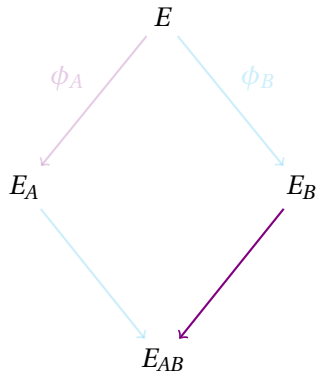
Attack framework

- Alice uses a **static** private key (m_A, n_A)
 $\implies E_A$ and ϕ_A can be precomputed



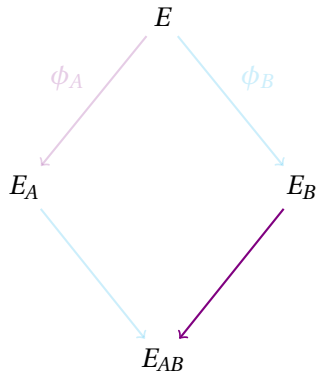
Attack framework

- Alice uses a **static** private key (m_A, n_A)
 $\implies E_A$ and ϕ_A can be precomputed
- The attacker plays the role of Bob



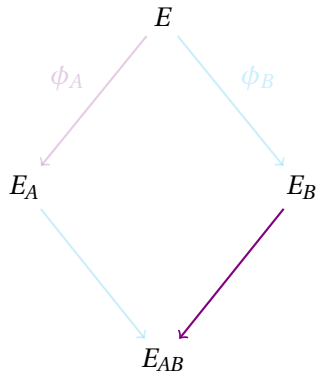
Attack framework

- Alice uses a **static** private key (m_A, n_A)
 $\implies E_A$ and ϕ_A can be precomputed
- The attacker plays the role of Bob
- Focus on the isogeny from E_B to $E_B / \langle m_A P'_A + n_A Q'_A \rangle$,
where $P'_A = \phi_B(P_A)$ and $Q'_A = \phi_B(Q_A)$



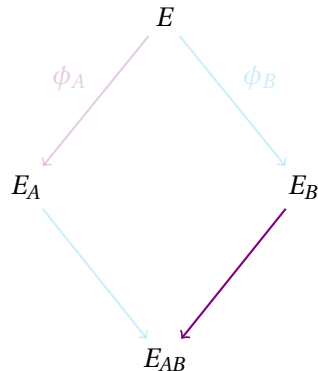
Attack framework

- Alice uses a **static** private key (m_A, n_A)
 $\implies E_A$ and ϕ_A can be precomputed
- The attacker plays the role of Bob
- Focus on the isogeny from E_B to $E_B / \langle m_A P'_A + n_A Q'_A \rangle$,
where $P'_A = \phi_B(P_A)$ and $Q'_A = \phi_B(Q_A)$
- Previous active attack [GPST16]:
 - **Idea:** Provide dishonest points $(\tilde{P}_A, \tilde{Q}_A)$ instead of (P'_A, Q'_A)



Attack framework

- Alice uses a **static** private key (m_A, n_A)
 $\Rightarrow E_A$ and ϕ_A can be precomputed
- The attacker plays the role of Bob
- Focus on the isogeny from E_B to $E_B / \langle m_A P'_A + n_A Q'_A \rangle$,
where $P'_A = \phi_B(P_A)$ and $Q'_A = \phi_B(Q_A)$
- Previous active attack [GPST16]:
 - **Idea**: Provide dishonest points $(\tilde{P}_A, \tilde{Q}_A)$ instead of (P'_A, Q'_A)
 - **Countermeasure**: Validation method verifies the correctness of the inputs (Fujisaki-Okamoto transform)



How is computed the isogeny ?

- Degree ℓ_A^n : Vélu's formulae $\Rightarrow O(\ell_A^n)$



How is computed the isogeny ?

- Degree ℓ_A^n : Vélu's formulae $\Rightarrow O(\ell_A^n)$



- Decompose and iterate $\Rightarrow n \cdot O(\ell_A)$



$$E_B = E_0 \rightarrow E_1 \rightarrow \dots \rightarrow E_{n-1} \rightarrow E_n = E_{AB}$$

where each \rightarrow is a degree- ℓ_A isogeny

How is computed the isogeny ?

- Degree ℓ_A^n : Vélu's formulae $\Rightarrow O(\ell_A^n)$ ☹️

- Decompose and iterate $\Rightarrow n \cdot O(\ell_A)$ 😊

$$E_B = E_0 \rightarrow E_1 \rightarrow \dots \rightarrow E_{n-1} \rightarrow E_n = E_{AB}$$

where each \rightarrow is a degree- ℓ_A isogeny

- $R_0 = m_A P'_A + n_A Q'_A$ and for $1 \leq k \leq n-1$,

$$E_{k+1} = E_k / \langle \ell_A^{n-k-1} R_k \rangle \quad \phi_{k+1} : E_k \rightarrow E_{k+1} \quad R_{k+1} = \phi_{k+1}(R_k)$$

How is computed the isogeny ?

- Degree ℓ_A^n : Vélu's formulae $\Rightarrow O(\ell_A^n)$



- Decompose and iterate $\Rightarrow n \cdot O(\ell_A)$



$$E_B = E_0 \rightarrow E_1 \rightarrow \dots \rightarrow E_{n-1} \rightarrow E_n = E_{AB}$$

where each \rightarrow is a degree- ℓ_A isogeny

- $R_0 = m_A P'_A + n_A Q'_A$ and for $1 \leq k \leq n-1$,

$$E_{k+1} = E_k / \langle \ell_A^{n-k-1} R_k \rangle \quad \phi_{k+1} : E_k \rightarrow E_{k+1} \quad R_{k+1} = \phi_{k+1}(R_k)$$

- $E_n = E_{AB} = E_B / \langle R_0 \rangle$ and $\phi = \phi_n \circ \dots \circ \phi_1$

Loop-abort fault attacks

- Introduced for pairing-based cryptography
Used recently in the context of lattice-based signature schemes

Loop-abort fault attacks

- Introduced for pairing-based cryptography
Used recently in the context of lattice-based signature schemes
- Inject a fault that induces an early-abort in the loop

Loop-abort fault attacks

- Introduced for pairing-based cryptography
Used recently in the context of lattice-based signature schemes
- Inject a fault that induces an early-abort in the loop
- Proven feasible in practice [Blömer *et al.*]

Loop-abort fault attacks

- Introduced for pairing-based cryptography
Used recently in the context of lattice-based signature schemes
- Inject a fault that induces an early-abort in the loop
- Proven feasible in practice [Blömer *et al.*]
- Implementations of SIDH on embedded devices already exist

- Need an oracle to compare Alice's outputs with what the attacker computes

- Need an oracle to compare Alice's outputs with what the attacker computes
- After k iterations, Alice has computed the intermediate curve

$$E_k = E_B / \langle 2^{n-k}(m_A P'_A + n_A Q'_A) \rangle$$

- Need an oracle to compare Alice's outputs with what the attacker computes
- After k iterations, Alice has computed the intermediate curve

$$E_k = E_B / \langle 2^{n-k}(m_A P'_A + n_A Q'_A) \rangle$$

- **Guess strategy:** first step, $k = 1$

- Need an oracle to compare Alice's outputs with what the attacker computes
- After k iterations, Alice has computed the intermediate curve

$$E_k = E_B / \langle 2^{n-k}(m_A P'_A + n_A Q'_A) \rangle$$

- **Guess strategy:** first step, $k = 1$
 - if m_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} Q'_A \rangle$

- Need an oracle to compare Alice's outputs with what the attacker computes
- After k iterations, Alice has computed the intermediate curve

$$E_k = E_B / \langle 2^{n-k}(m_A P'_A + n_A Q'_A) \rangle$$

- **Guess strategy:** first step, $k = 1$
 - if m_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} Q'_A \rangle$
 - if n_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} P'_A \rangle$

- Need an oracle to compare Alice's outputs with what the attacker computes
- After k iterations, Alice has computed the intermediate curve

$$E_k = E_B / \langle 2^{n-k}(m_A P'_A + n_A Q'_A) \rangle$$

- **Guess strategy:** first step, $k = 1$
 - if m_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} Q'_A \rangle$
 - if n_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} P'_A \rangle$
 - if both are odd, then $\text{Ker } \phi_1 = \langle 2^{n-1}(P'_A + Q'_A) \rangle$

- Need an oracle to compare Alice's outputs with what the attacker computes
- After k iterations, Alice has computed the intermediate curve

$$E_k = E_B / \langle 2^{n-k}(m_A P'_A + n_A Q'_A) \rangle$$

- **Guess strategy:** first step, $k = 1$
 - if m_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} Q'_A \rangle$
 $\implies (m_A, n_A)$ equivalent to $(a, 1)$ for $a = \frac{m_A}{n_A}$ and a even
 - if n_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} P'_A \rangle$
 - if both are odd, then $\text{Ker } \phi_1 = \langle 2^{n-1}(P'_A + Q'_A) \rangle$

- Need an oracle to compare Alice's outputs with what the attacker computes
- After k iterations, Alice has computed the intermediate curve

$$E_k = E_B / \langle 2^{n-k}(m_A P'_A + n_A Q'_A) \rangle$$

- **Guess strategy:** first step, $k = 1$
 - if m_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} Q'_A \rangle$
 $\implies (m_A, n_A)$ equivalent to $(a, 1)$ for $a = \frac{m_A}{n_A}$ and a even
 - if n_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} P'_A \rangle$
 $\implies (m_A, n_A)$ equivalent to $(1, a)$ for $a = \frac{n_A}{m_A}$ and a even
 - if both are odd, then $\text{Ker } \phi_1 = \langle 2^{n-1}(P'_A + Q'_A) \rangle$

- Need an oracle to compare Alice's outputs with what the attacker computes
- After k iterations, Alice has computed the intermediate curve

$$E_k = E_B / \langle 2^{n-k}(m_A P'_A + n_A Q'_A) \rangle$$

- **Guess strategy:** first step, $k = 1$
 - if m_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} Q'_A \rangle$
 $\implies (m_A, n_A)$ equivalent to $(a, 1)$ for $a = \frac{m_A}{n_A}$ and a even
 - if n_a is even, then $\text{Ker } \phi_1 = \langle 2^{n-1} P'_A \rangle$
 $\implies (m_A, n_A)$ equivalent to $(1, a)$ for $a = \frac{n_A}{m_A}$ and a even
 - if both are odd, then $\text{Ker } \phi_1 = \langle 2^{n-1}(P'_A + Q'_A) \rangle$
 $\implies (m_A, n_A)$ equivalent to $(1, a)$ for $a = \frac{n_A}{m_A}$ and a odd

- **Subsequent steps:** we assume the key of the form $(1, a)$

- **Subsequent steps:** we assume the key of the form $(1, a)$
- We know the $k-1$ least significant bits

- **Subsequent steps:** we assume the key of the form $(1, a)$
- We know the $k-1$ least significant bits
- The k -th bit is either 0 or 1, *i.e.*,

$$E_k = E_B / \left\langle 2^{n-k} \left(P'_A + (a \bmod 2^{k-1}) Q'_A \right) \right\rangle \quad \text{or} \quad E_k = E_B / \left\langle 2^{n-k} \left(P'_A + (a \bmod 2^{k-1} + 2^{k-1}) Q'_A \right) \right\rangle$$

- **Subsequent steps:** we assume the key of the form $(1, a)$
- We know the $k-1$ least significant bits
- The k -th bit is either 0 or 1, *i.e.*,

$$E_k = E_B / \left\langle 2^{n-k} \left(P'_A + (a \bmod 2^{k-1}) Q'_A \right) \right\rangle \quad \text{or} \quad E_k = E_B / \left\langle 2^{n-k} \left(P'_A + (a \bmod 2^{k-1} + 2^{k-1}) Q'_A \right) \right\rangle$$

- Make a guess and recover the k -th bit of a

- **Subsequent steps:** we assume the key of the form $(1, a)$
- We know the $k-1$ least significant bits
- The k -th bit is either 0 or 1, *i.e.*,

$$E_k = E_B / \left\langle 2^{n-k} \left(P'_A + (a \bmod 2^{k-1}) Q'_A \right) \right\rangle \quad \text{or} \quad E_k = E_B / \left\langle 2^{n-k} \left(P'_A + (a \bmod 2^{k-1} + 2^{k-1}) Q'_A \right) \right\rangle$$

- Make a guess and recover the k -th bit of a
- **Conclusion:** full-key recovery by iterating this process

- n bits recovered in n interactions with the victim $\implies n$ faults injected

- n bits recovered in n interactions with the victim $\implies n$ faults injected
- if the success probability μ of the fault injection is not 1,

- n bits recovered in n interactions with the victim $\implies n$ faults injected
- if the success probability μ of the fault injection is not 1,
 - about $\frac{n}{\mu}$ faults injected if the success can be detected

- n bits recovered in n interactions with the victim $\implies n$ faults injected
- if the success probability μ of the fault injection is not 1,
 - about $\frac{n}{\mu}$ faults injected if the success can be detected
 - about $\frac{2n}{\mu}$ otherwise

- n bits recovered in n interactions with the victim $\implies n$ faults injected
- if the success probability μ of the fault injection is not 1,
 - about $\frac{n}{\mu}$ faults injected if the success can be detected
 - about $\frac{2n}{\mu}$ otherwise
- Alternative with less faults assuming a stronger oracle

Bedankt