Implementation and Evaluation of a Leakage-Resilient ElGamal KEM

David Galindo¹,², Johann Großschädl³, Zhe Liu³, Praveen K. Vadnala³, Srinivas Vivek³

¹ CNRS/Loria, France  
² SCYTL Secure Electronic Voting, Spain  
³ University of Luxembourg

PROOFS 2014
Side-Channel Attacks

Use data leaked due to the physical nature of computation:

- running time
- power consumption
- electromagnetic-radiation leak
- acoustic emanation
- photons emissions
- ground electric potential
- fault attacks
Side-Channel Attacks Countermeasures

Aimed at specific attacks
Concrete implementations
Leakage model meaningful
Reasonably practical
SCA-resistant primitives

SCA Countermeasures flow

input message
\( K^* \)
target computation
\( f(K^*, T) \)
leakage model
\( \varphi \)
noise
\( N \)
actual leakage
\( X \approx \varphi((K^*, T)) \)
distinguisher
\( D \)
attack/non-attack
\( \hat{K} = D(X, T) \)
Side-Channel Attacks Countermeasures

Aimed at specific attacks
Concrete implementations
Leakage model meaningful
Reasonably practical
SCA-resistant primitives

However...

SCA Countermeasures

flow

input message
$K^*$
target computation
$f(K^*, T)$
leakage model
$\varphi$
noise
$N$
actual leakage
$X \approx N(\varphi((K^*, T)))$
distinguisher
$D$
attack/non-attack
$\hat{K} = D(X, T)$

David Galindo – SCYTL Secure Electronic Voting
Evaluation of a Leakage-Resilient ElGamal KEM
Side-Channel Attacks Countermeasures

- Aimed at specific attacks
- Concrete implementations
- Leakage model meaningful
- Reasonably practical
- SCA-resistant primitives

A new attack \((\varphi, \mathbf{N}, \mathcal{D})\) might be discovered

- Endless? **cat-and-mouse** game

### SCA Countermeasures flow

**input message**
\[ K^* \]

**target computation**
\[ f(K^*, T) \]

**leakage model**
\[ \varphi \]

**noise**
\[ \mathbf{N} \]

**actual leakage**
\[ \mathbf{X} \approx N \varphi((K^*, T)) \]

**distinguisher**
\[ \mathcal{D} \]

**security?**
\[ \hat{K} = \mathcal{D}(\mathbf{X}, T) \]
## SCA Countermeasures vs. Leakage-Resilient Cryptography

<table>
<thead>
<tr>
<th>SCA countermeasures</th>
<th>Leakage-Resilient Crypto</th>
</tr>
</thead>
<tbody>
<tr>
<td>😞 Aimed at specific attacks</td>
<td>😊 Aimed at generic attacks</td>
</tr>
<tr>
<td>😊 Concrete implementations</td>
<td>😞 No implementations</td>
</tr>
<tr>
<td>😊 Leakage model meaningful</td>
<td>😞 Leakage model generic</td>
</tr>
<tr>
<td>😊 Reasonably practical SCA-resistant primitives</td>
<td>😞 Not practical</td>
</tr>
<tr>
<td>A new attack ((\varphi, N, D)) might be discovered</td>
<td>😊 Security reduction</td>
</tr>
<tr>
<td>😞 Endless? cat-and-mouse game</td>
<td></td>
</tr>
</tbody>
</table>
In this work we take a step forward towards this goal.

- Aimed at general attacks
- Leakage model meaningful
- Reasonably practical SCA-resistant primitives
- Security reduction
- Concrete implementations
Meaningful Leakage-Resilient Cryptography

- Aimed at general attacks
- Leakage model meaningful
- Reasonably practical SCA-resistant primitives
- Security reduction
- Concrete implementations

In this work we take a step forward towards to this goal
Our contribution

- **A more reasonable** leakage modeling
- We depart from an existing practical ElGamal KEM and modify it using **practical motivations**
- We use the **theory and practice** of SCA to argue that it potentially meets the leakage bound
- We **implement** the scheme on an ARM Cortex M-3 processor
A **stateful** KEM scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}_1, \text{Dec}_2)$ consists of efficient algorithms:

- $\text{KeyGen}(\lambda)$ outputs $(pk, (sk_0, sk'_0))$
- $\text{Enc}(pk)$ outputs $(K, C)$
- $\text{Dec}_1(sk_{i-1}, C)$ updates $sk_{i-1}$ to $sk_i$ and outputs intermediate state $w_i$
- $\text{Dec}_2(sk'_{i-1}, w_i)$ updates $sk'_{i-1}$ to $sk'_i$ and outputs key $K$ or $\perp$
ElGamal KEM with Multiplicative Masking

- **KG(κ):** choose $x, t_0 \leftarrow \mathbb{Z}_q$. Set $X = g^x$, $sk_0 = t_0$, $sk'_0 = x/t_0$. Return $(X, (sk_0, sk'_0))$

- **Enc(pk):** choose $r \leftarrow \mathbb{Z}_q$. Compute $C = g^r$ and $K = X^r$; return $(C, K)$

- **Dec1(sk_{i-1}, C):** pick $t_i \leftarrow \mathbb{Z}_q$, set $sk_i = sk_{i-1} \cdot t_i$, $Y_i = C^{sk_i}$. Return $(t_i, Y_i)$

- **Dec2(sk'_{i-1}, (t_i, Y_i), C):** set $sk'_i = sk'_{i-1} \cdot t_i^{-1}$, and return $K = Y_i^{sk'_i}$. 

We consider chosen-ciphertext and leakage security against lunch-time attacks (CCLA1)

CCLA1 Experiment

\[
\text{KEM-CCLA1}_\text{KEM}(\mathcal{A}, \kappa, \lambda) \\
(pk, (sk_0, sk'_0)) \leftarrow \text{KG}^*(\kappa, \lambda) \\
w \leftarrow \mathcal{A}^{\text{O}^{\text{CCLA1}}}(\cdot)(pk) \\
b \leftarrow \{0, 1\} \\
(C^*, K_0) \leftarrow \text{Enc}^*(pk) \\
K_1 \leftarrow \mathcal{K} \\
b' \leftarrow \mathcal{A}(w, C^*, K_b)
\]

\[
\text{KEM-Leak-Oracle} \quad O^{\text{CCLA1}}(C, f_i, h_i) \\
(sk_i, w_i) \leftarrow \text{Dec}^1*(sk_{i-1}, C) \\
(sk'_i, K) \leftarrow \text{Dec}^2*(sk'_{i-1}, w_i) \\
\Lambda_i := f_i(sk_{i-1}, r_i) \\
\Lambda'_i := h_i(sk'_{i-1}, r'_i, w_i) \\
i := i + 1 \\
\text{Return } (K, \Lambda_i, \Lambda'_i)
\]
We consider chosen-ciphertext and leakage security against lunch-time attacks (CCLA1)

CCLA1 Experiment

\[ \mathsf{KEM-CCLA1}_{\mathsf{KEM}}(A, \kappa, \lambda) \]
\[ (pk, (sk_0, sk'_0)) \leftarrow \mathsf{KG}^*(\kappa, \lambda) \]
\[ w \leftarrow A^{\mathsf{O_{CCLA1}}}(pk) \]
\[ b \leftarrow \{0, 1\} \]
\[ (C^*, K_0) \leftarrow \mathsf{Enc}^*(pk) \]
\[ K_1 \leftarrow \mathcal{K} \]
\[ b' \leftarrow A(w, C^*, K_b) \]

\[ \mathsf{KEM-Leak-Oracle}^{\mathsf{O_{CCLA1}}}(C, f_i, h_i) \]
\[ (sk_i, w_i) \leftarrow \mathsf{Dec}^1(\mathsf{sk}_{i-1}, C) \]
\[ (sk'_i, K) \leftarrow \mathsf{Dec}^2(\mathsf{sk}'_{i-1}, w_i) \]
\[ \Lambda_i := f_i(\mathsf{sk}_{i-1}, r_i) \]
\[ \Lambda'_i := h_i(\mathsf{sk}'_{i-1}, r'_i, w_i) \]
\[ i := i + 1 \]
Return \( (K, \Lambda_i, \Lambda'_i) \)

Restriction on leakage functions \( f_i, h_i \)

\[ \tilde{H}_\infty (t \mid f_i(\sigma_{i-1}, r_i)) \geq H_\infty (t) - \lambda \quad \forall t \in \sigma_{i-1} \cup r_i, \]
\[ \tilde{H}_\infty (t \mid h_i(\sigma'_{i-1}, r'_i, w_i)) \geq H_\infty (t) - \lambda \quad \forall t \in \sigma'_{i-1} \cup r'_i \cup w_i. \]
State of the art does not allow to give a security reduction with leakage

If $f_i, h_i$ leak $\lambda \geq 3/8 \log q$ bits of each share of the secret key, then there exists a heuristic attack [Galindo-Vivek, IPL 2014]

Probably due to the fact that any exponentiation algorithm inherently leaks information about the exponent
State of the art does not allow to give a security reduction with leakage

If $f_i, h_i$ leak $\lambda \geq 3/8 \log q$ bits of each share of the secret key, then there exists a heuristic attack [Galindo-Vivek, IPL 2014]

Probably due to the fact that any exponentiation algorithm inherently leaks information about the exponent

Idea! Avoid placing secret data on your exponentiations’ exponents...
Asymmetric Pairings

- Let $G_1, G_2, G_T$ be groups of prime order $q$
- $G_1 = < g >, G_2 = < G >$
- Pairing $e : G_1 \times G_2 \rightarrow G_T$
  - bilinear: $e(g^a, g^b) = e(g, g)^{ab}, \forall a, b \in \mathbb{Z}$
  - non-degenerate: $G_T = < e(g, G) >$
Pairing-Based Stateful ElGamal KEM (Asiacrypt 2010)

- **KG(κ)**: choose $x, t_0 \xleftarrow{\$} \mathbb{Z}_q$. Set $X = g^x, sk_0 = g^{t_0}, sk'_0 = g^{x-t_0}$, and $X_T = e(X, G)$. Return $(X_T, (sk_0, sk'_0))$

- **Enc(pk)** choose $r \xleftarrow{\$} \mathbb{Z}_q$. Compute $C = G^r$ and $K = X_T^r$; return $(C, K)$

- **Dec1(C, sk_{i-1})** pick $t_i \xleftarrow{\$} \mathbb{Z}_q$, set $sk_i = sk_{i-1} \cdot G^{t_i}$, $Y_i = e(sk_i, C)$. Return $(t_i, Y_i)$

- **Dec2(sk'_{i-1}, (t_i, Y_i), C)** set $sk'_i = sk'_{i-1} \cdot G^{-t_i}$, and $Y'_i = e(sk'_i, C)$. Return $K = Y_i \cdot Y'_i \in G_T$
ElGamal KEM with Multiplicative Masking

- **KG(κ):** choose $x, t_0 \leftarrow \mathbb{Z}_q$. Set $X = g^x$, $sk_0 = t_0$, $sk'_0 = x/t_0$. Return $(X, (sk_0, sk'_0))$

- **Enc(pk):** choose $r \leftarrow \mathbb{Z}_q$. Compute $C = g^r$ and $K = X^r$; return $(C, K)$

- **Dec1(sk_{i-1}, C):** pick $t_i \leftarrow \mathbb{Z}_q$, set $sk_i = sk_{i-1} \cdot t_i$, $Y_i = C^{sk_i}$. Return $(t_i, Y_i)$

- **Dec2(sk'_{i-1}, (t_i, Y_i), C):** set $sk'_i = sk'_{i-1} \cdot t_i^{-1}$, and return $K = Y_i^{sk'_i}$. 

David Galindo – SCYTL Secure Electronic Voting

Evaluation of a Leakage-Resilient ElGamal KEM
Pairing-Based Stateful ElGamal KEM (Asiacrypt 2010)

- **KG(κ):** choose $x, t_0 \leftarrow \mathbb{Z}_q$. Set $X = g^x$, $sk_0 = g^{t_0}$, $sk'_0 = g^{x-t_0}$, and $X_T = e(X, G)$. Return $(X_T, (sk_0, sk'_0))$

- **Enc(pk):** choose $r \leftarrow \mathbb{Z}_q$. Compute $C = G^r$ and $K = X_T^r$; return $(C, K)$

- **Dec1(C, sk_{i-1}):** pick $t_i \leftarrow \mathbb{Z}_q$, set $sk_i = sk_{i-1} \cdot G^{t_i}$, $Y_i = e(sk_i, C)$. Return $(t_i, Y_i)$

- **Dec2(sk'_{i-1}, (t_i, Y_i), C):** set $sk'_i = sk'_{i-1} \cdot G^{-t_i}$, and $Y'_i = e(sk'_i, C)$. Return $K = Y_i \cdot Y'_i \in \mathbb{G}_T$

Security reduction in the Generic Bilinear Group Model if the leakage is bounded in size
**Pairing-Based Stateful ElGamal KEM (Asiacrypt 2010)**

- **KG(κ):** choose \( x, t_0 \leftarrow Z_q \). Set \( X = g^x, sk_0 = g^{t_0}, sk'_0 = g^{x-t_0} \), and \( X_T = e(X, G) \). Return \( (X_T, (sk_0, sk'_0)) \)

- **Enc(pk):** choose \( r \leftarrow Z_q \). Compute \( C = G^r \) and \( K = X_T^r \); return \( (C, K) \)

- **Dec1(C, sk_{i-1}):** pick \( t_i \leftarrow Z_q \), set \( sk_i = sk_{i-1} \cdot G^{t_i} \), \( Y_i = e(sk_i, C) \). Return \( (t_i, Y_i) \)

- **Dec2(sk'_{i-1}, (t_i, Y_i), C):** set \( sk'_i = sk'_{i-1} \cdot G^{-t_i} \), and \( Y'_i = e(sk'_i, C) \). Return \( K = Y_i \cdot Y'_i \in G_T \)

Security reduction in the Generic Bilinear Group Model if the leakage is **bounded in size**

Non-meaningful leakage model...
Pairing-Based Stateful ElGamal KEM (Asiacrypt 2010)

- **KG(κ):** choose \( x, t_0 \leftarrow Z_q \). Set \( X = g^x \), \( sk_0 = g^{t_0} \), \( sk'_0 = g^{x-t_0} \), and \( X_T = e(X, G) \). Return \((X_T, (sk_0, sk'_0))\)

- **Enc(pk):** choose \( r \leftarrow Z_q \). Compute \( C = G^r \) and \( K = X_T^r \); return \((C, K)\)

- **Dec1(C, sk_{i-1}):** pick \( t_i \leftarrow Z_q \), set \( sk_i = sk_{i-1} \cdot G^{t_i} \), \( Y_i = e(sk_i, C) \). Return \((t_i, Y_i)\)

- **Dec2(sk'_{i-1}, (t_i, Y_i), C):** set \( sk'_i = sk'_{i-1} \cdot G^{-t_i} \), and \( Y'_i = e(sk'_i, C) \). Return \( K = Y_i \cdot Y'_i \in \mathbb{G}_T\)

We did not get rid of exponentiations that place secret data on the exponent...
KG(κ): choose $x, t_0 \leftarrow \mathbb{Z}_q$. Set $X = g^x$, $sk_0 = g^{t_0}$, $sk'_0 = g^{x-t_0}$, and $X_T = e(X, G)$. Return $(X_T, (sk_0, sk'_0))$

Enc(pk) choose $r \leftarrow \mathbb{Z}_q$. Compute $C = G^r$ and $K = X_T^r$; return $(C, K)$

Dec1($C, sk_{i-1}$) pick $U_i \leftarrow \mathbb{G}_1$, set $sk_i = sk_{i-1} \cdot U_i$, $Y_i = e(sk_i, C)$. Return $(U_i, Y_i)$

Dec2($sk'_{i-1}, (U_i, Y_i), C$) set $sk'_i = sk'_{i-1} \cdot U^{-1}$, and $Y'_i = e(sk'_i, C)$. Return $K = Y_i \cdot Y'_i \in \mathbb{G}_T$

Look, there is no need to exponentiate...
Computing random \( u_i = g^{t_i} \) for \( t_i \in \mathbb{F}_q \) leaks information on the fresh randomness used for decryption.

We do not know any exponentiation algorithm susceptible to meet the leakage bound.

We do not need knowledge of \( t_i = \log_g u_i \).

We use an encoding \( f : \mathbb{F}_p \mapsto E(\mathbb{F}_p) \) with good randomness preserving properties.

This encoding is naturally almost leakage-free.
BEG-KEM+

KG$^+_\text{BEG}(\kappa)$ choose $x, t_0 \leftarrow \mathbb{Z}_q$. Set $X = g^x, sk_0 = g^{t_0}, sk'_0 = g^{x-t_0}$, and $X_T = e(X, G)^x$. Return $(X_T, (sk_0, sk'_0))$

Enc$^+_\text{BEG}(pk)$ choose $r \leftarrow \mathbb{Z}_q$, compute $C = G^r$ and $K = X_T^r$

Dec1$^+_\text{BEG}(sk_{i-1}, C)$ choose $t_i, z_i \leftarrow \mathbb{F}_p$, set $u_i = f(t_i) \cdot f(z_i)$, and compute $sk_i = sk_{i-1} \cdot u_i$ and $Y_i = e(sk_i, C)$. Return $(u_i, Y_i)$

Dec2$^+_\text{BEG}(sk'_{i-1}, (u_i, Y_i), C)$ Set $sk'_i = sk'_{i-1} \cdot (u_i)^{-1}$ and $Y'_i = e(sk'_i, C)$. Return $K = Y_i \cdot Y'_i \in \mathbb{G}_T$
Fouque-Tibouchi encoding to Barreto-Naehrig curves

**Require:** A random number \( t \in \mathbb{F}_p \)

**Ensure:** Point \( P \in E(\mathbb{F}_p) \)

1. \( w \leftarrow \sqrt{-3} \cdot t / (1 + b + t^2) \)
2. \( x_1 \leftarrow (-1 + \sqrt{-3}) / 2 - tw \)
3. \( x_2 \leftarrow -1 - x_1 \)
4. \( x_3 \leftarrow 1 + 1/w^2 \)
5. \( r_1, r_2, r_3 \leftarrow \mathbb{F}_q^* \)
6. \( \alpha \leftarrow \chi_p(r_1^2 \cdot (x_1^3 + b)) \)
7. \( \beta \leftarrow \chi_p(r_2^2 \cdot (x_2^3 + b)) \)
8. \( i \leftarrow [(\alpha - 1) \cdot \beta \mod 3] + 1 \)
9. return \( P[x_i, \chi_p(r_3^2 \cdot t) \cdot \sqrt{(x_i^3 + b)}] \)

- \( p \equiv 3 \mod 4 \)
  - \( \chi_p(\cdot) \) is the Legendre symbol
- Use Extended Euclidean Algo to compute inverses as:
  \[ \frac{1}{x} = \frac{1}{x \cdot r} \cdot r \text{ for } r \leftarrow \mathbb{F}_p \]
- \( \sqrt{x} \) for \( x \in \mathbb{F}_p \) is computed as a fixed-exponent computation:
  \[ \sqrt{x} = x^{p+1}/4 \]
Fouque-Tibouchi encoding to Barreto-Naehrig curves

Require: A random number \( t \in \mathbb{F}_p \)

Ensure: Point \( P \in E(\mathbb{F}_p) \)

1: \( w \leftarrow \sqrt{-3} \cdot t / (1 + b + t^2) \)
2: \( x_1 \leftarrow (-1 + \sqrt{-3})/2 - tw \)
3: \( x_2 \leftarrow -1 - x_1 \)
4: \( x_3 \leftarrow 1 + 1/w^2 \)
5: \( r_1, r_2, r_3 \leftarrow \mathbb{F}_q^* \)
6: \( \alpha \leftarrow \chi_p(r_1^2 \cdot (x_1^3 + b)) \)
7: \( \beta \leftarrow \chi_p(r_2^2 \cdot (x_2^3 + b)) \)
8: \( i \leftarrow [(\alpha - 1) \cdot \beta \mod 3] + 1 \)
9: return \( P[x_i, \chi_p(r_3^2 \cdot t) \cdot \sqrt{(x_i^3 + b)}] \)

- \( p \equiv 3 \mod 4 \)
  - \( \chi_p(\cdot) \) is the Legendre symbol

- Use Extended Euclidean Algo to compute inverses as:
  \( \frac{1}{x} = \frac{1}{x \cdot r} \cdot r \)
  for \( r \leftarrow \mathbb{F}_p \)

- \( \sqrt{x} \) for \( x \in \mathbb{F}_p \) is computed as a fixed-exponent computation:
  \( \sqrt{x} = x^{\frac{p+1}{4}} \)

There are no branching instructions in the computation of the encoding
We present a security reduction in the Generic Bilinear Group Model if the leakage is **does not decrease** the min-entropy of the (intermediate) secret values “too much”...
We present a security reduction in the Generic Bilinear Group Model if the leakage is **does not decrease** the min-entropy of the (intermediate) secret values “too much”...

par single trace!

Great bonus: attacks that require multiple traces are **ruled out**

Michael Scott in [Computing the Tate pairing, CT-RSA 2005] claims:

"One might with reasonable confidence expect that the power consumption profile of (and execution time for) such protocols [against SPA attacks] will be constant and independent of any secret values."
[Unterluggauer-Wenger, ARES 2014] CPA attack with 1500 traces in an ARM Cortex-M0 processor


no attacks known with single (or few) trace(s)!
Pairings and Single Trace Attacks

[Unterluggauer-Wenger, ARES 2014] CPA attack with 1500 traces in an ARM Cortex-M0 processor


no attacks known with single (or few) trace(s)!

- “Intrinsically” more secure than e.g. exponentiation since the critical input data is a secret group element and not a secret scalar
- Operand-related SPA leakage from field-arithmetic operations is generally small (in large characteristic)
Implementation

- Barreto-Naehrig curve defined over a 254-bit prime field $\mathbb{F}_p$
- We implemented BEG-KEM+ in ANSI C
- MIRACL library for an efficient execution of the pairing evaluation
- Adruino Due microcontroller board with an ARM Cortex-M3 CPU

Table: Running times in $10^6$ clock cycles

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square root $\mathbb{F}_p$</td>
<td>0.7</td>
</tr>
<tr>
<td>Inversion $\mathbb{F}_p$</td>
<td>0.087</td>
</tr>
<tr>
<td>Encoding to $\mathbb{G}_2$</td>
<td>3.7</td>
</tr>
<tr>
<td>Exponentiation $\mathbb{G}_1$</td>
<td>4.5</td>
</tr>
<tr>
<td>Exponentiation $\mathbb{G}_2$</td>
<td>10.0</td>
</tr>
<tr>
<td>Exponentiation $\mathbb{G}_T$</td>
<td>27.1</td>
</tr>
<tr>
<td>Pairing</td>
<td>65.0</td>
</tr>
</tbody>
</table>

Table: Comparison of BEG-KEM and BEG-KEM+

<table>
<thead>
<tr>
<th>Operation</th>
<th>BEG-KEM</th>
<th>BEG-KEM+</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen</td>
<td>108</td>
<td>108</td>
</tr>
<tr>
<td>Encryption</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>Decryption</td>
<td>131</td>
<td>140</td>
</tr>
</tbody>
</table>
Conclusions

- We (would have liked to) contribute to bridge approaches for SCA resistance
  - SCA practice & countermeasures
  - provable security
- We provided a more reasonable leakage modeling
- We present a scheme and argue that it is susceptible to meet the leakage requirement
- We provided an implementation in an ARM Cortex-M3 processor
- Pairings have proven to be very useful in multiple contexts
  Maybe also for building SCA-resistant implementations?
- We continue exploring this approach
That's all folks! 😊