Masking the GLP Lattice-Based Signature Scheme at any Order

Gilles Barthe (IMDEA Software Institute)
Sonia Belaïd (CryptoExperts)
Thomas Espitau (UPMC)
Pierre-Alain Fouque (Univ. Rennes I and IUF)
Benjamin Grégoire (INRIA Sophia Antipolis)
Mélissa Rossi (ENS Paris and Thales)
Mehdi Tibouchi (NTT Secure Platform Laboratories)

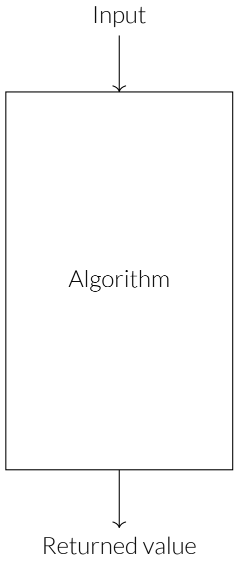
May 1st 2018 Eurocrypt

Masking a post-quantum signature

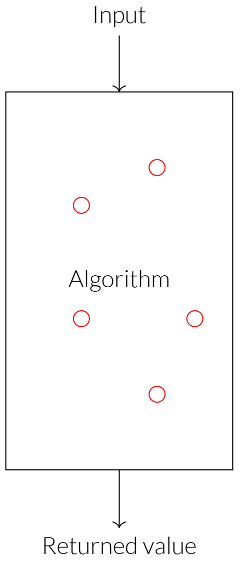
- Numerous side channel attacks against lattice-based schemes (Gaussian distributions, rejection sampling)
- Few countermeasures exist, especially on signatures
- Call for concrete implementations of post-quantum cryptography

Strong countermeasures needed

Leakage models and masking



Leakage models and masking



Ishai, Sahai and Wagner model [ISW03] :
The attacker can access the exact values of at most d intermediate values

Proof-Friendly

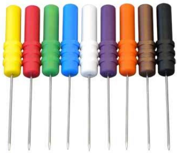
Leakage models and masking

Input

Algorithm

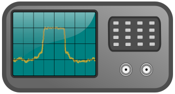
Returned value

Ishai, Sahai and Wagner model [ISW03] :
The attacker can access the exact values of at most d intermediate values



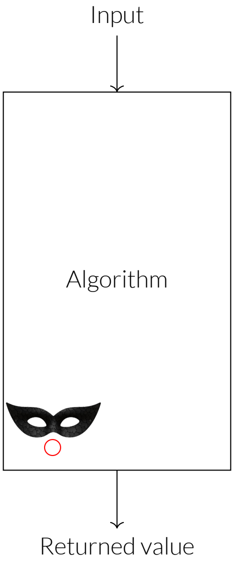
Proof-Friendly

Noisy leakage model [CJRR99, PR13]:
The attacker can access the **noisy** values of all the intermediate values

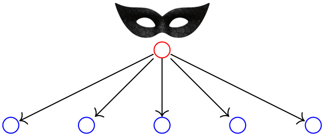


Realistic

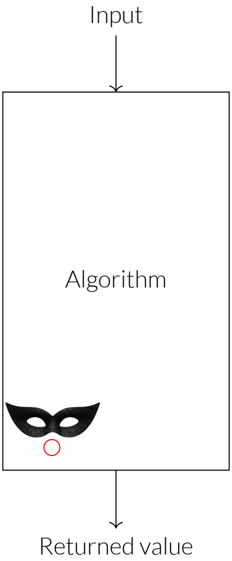
Leakage models and masking



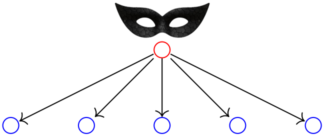
Security in the ISW model: d order **masking**
Each sensitive value is replaced by $d+1$ shares.



Leakage models and masking

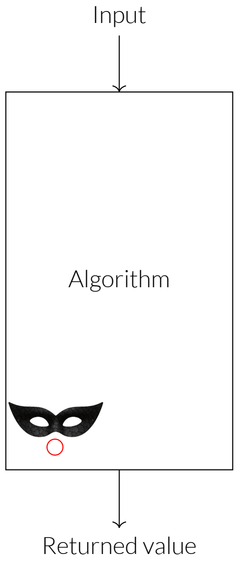


Security in the ISW model: d order **masking**
Each sensitive value is replaced by $d+1$ shares.



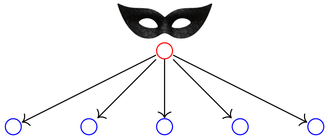
Such that it is impossible to recover the value without having all $d+1$ shares

Leakage models and masking



Security in the ISW model: d order **masking**

Each sensitive value is replaced by $d+1$ shares.



Such that it is impossible to recover the value without having all $d+1$ shares

Any strict subset of at most d shares is independant from the sensitive value

Our contribution

The first **provable** masked implementation of a lattice-based signature scheme **at any order**

- New techniques for masking lattice-based Fiat–Shamir with abort signatures
- New proofs for masking probabilistic algorithms

Why GLP signature scheme ?

Introduced in [Lyu09, Lyu12]
Implemented by **Güneysu, Lyubashevsky and Pöppelmann** in [GLP12]

- Ancestor of BLISS and Dilithium
- No Gaussians, only uniform distributions

Why GLP signature scheme ?

Introduced in [Lyu09, Lyu12]
Implemented by **Güneysu, Lyubashevsky and Pöppelmann** in [GLP12]

- Ancestor of BLISS and Dilithium
- No Gaussians, only uniform distributions

But still some new difficulties

- Probabilistic algorithm
- Reliance on rejection sampling

GLP Key derivation

$$\mathcal{R} = \frac{\mathbb{Z}_p[x]}{(x^n+1)}$$

$\mathcal{R}_k$: coefficients in the range $[-k, k]$

Algorithm 1 GLP key derivation

Ensure: Signing key $\boldsymbol{sk}$, verification key $\boldsymbol{pk}$

1: $\mathrm{s}_1, \mathrm{s}_2 \xleftarrow{\$} \mathcal{R}_1$ //s_1 and s_2 have coefficients in $\{-1, 0, 1\}$
2: $\mathrm{a} \xleftarrow{\$} \mathcal{R}$
3: $\mathrm{t} \leftarrow \mathrm{a}\mathrm{s}_1 + \mathrm{s}_2$
4: $\boldsymbol{sk} \leftarrow (\mathrm{s}_1, \mathrm{s}_2)$
5: $\boldsymbol{pk} \leftarrow (\mathrm{a}, \mathrm{t})$

- Based on the Decisional Compact Knapsack problem

GLP signature

- Fiat–Shamir with abort signature

Algorithm 2 GLP sign

Require: $m, pk = (a, t), sk = (s_1, s_2)$
Ensure: Signature σ

1: $y_1, y_2 \xleftarrow{\$} \mathcal{R}_k$ — Random generation
2: $c \leftarrow H(r = ay_1 + y_2, m)$ — Commitment and challenge
3: $z_1 \leftarrow s_1 c + y_1$
4: $z_2 \leftarrow s_2 c + y_2$
5: if z_1 or $z_2 \notin \mathcal{R}_{k-\alpha}$ then restart — Rejection Sampling
6: return $\sigma = (z_1, z_2, c)$

$$k = 2^{14} \quad \alpha = 16 \quad n = 512 \quad p = 8383489$$

Verification : $z_1, z_2 \in \mathcal{R}_{k-\alpha}$ and $c = H(az_1 + z_2 - tc, m)$

Structure of the countermeasure and its proof

1. The signature and key derivation algorithms are divided in blocks

Structure of the countermeasure and its proof

1. The signature and key derivation algorithms are divided in blocks
2. Each block is proven securely masked with one of the following properties

Structure of the countermeasure and its proof

1. The signature and key derivation algorithms are divided in blocks
2. Each block is proven securely masked with one of the following properties

Unmasked
For non sensitive parts.

Structure of the countermeasure and its proof

1. The signature and key derivation algorithms are divided in blocks
2. Each block is proven securely masked with one of the following properties

Unmasked
For non sensitive parts.

Non interferent
Every set of at most d intermediate variables can be perfectly simulated with at most d shares of each input.

Structure of the countermeasure and its proof

1. The signature and key derivation algorithms are divided in blocks
2. Each block is proven securely masked with one of the following properties

Unmasked
For non sensitive parts.

Non interferent
Every set of at most d intermediate variables can be perfectly simulated with at most d shares of each input.

Non interferent with public outputs
Every set of at most d intermediate variables can be perfectly simulated **with the public outputs and** at most d shares of each input.

New

Structure of the countermeasure and its proof

1. The signature and key derivation algorithms are divided in blocks
2. Each block is proven securely masked with one of the following properties

Unmasked
For non sensitive parts.

Non interferent
Every set of at most d intermediate variables can be perfectly simulated with at most d shares of each input.

Non interferent with public outputs
Every set of at most d intermediate variables can be perfectly simulated **with the public outputs and** at most d shares of each input.

We give some values (called outputs) to the attacker and prove that the countermeasure **does not leak more** than the outputs.

Structure of the countermeasure and its proof

1. The signature and key derivation algorithms are divided in blocks
2. Each block is proven securely masked with one of the following properties

Unnmasked
For non sensitive parts.

Non interferent
Every set of at most d intermediate variables can be perfectly simulated with at most d shares of each input.

Non interferent with public outputs
Every set of at most d intermediate variables can be perfectly simulated **with the public outputs and** at most d shares of each input.

We give some values (called outputs) to the attacker and prove that the countermeasure **does not leak more** than the outputs.

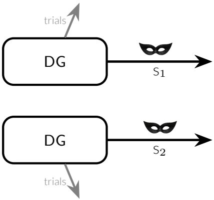
3. A composition proof combines all the securities to the whole scheme

Masking GLP key generation

Algorithm 1 GLP key generation

Ensure: Signing key $\boldsymbol{sk}$, verification key $\boldsymbol{pk}$

1: $\mathsf{s}_1, \mathsf{s}_2 \xleftarrow{\$} \mathcal{R}_1$ //s_1 and s_2 have coefficients in $\{-1, 0, 1\}$
2: $\mathsf{a} \xleftarrow{\$} \mathcal{R}$
3: $\mathsf{t} \leftarrow \mathsf{a}\mathsf{s}_1 + \mathsf{s}_2$
4: $\boldsymbol{sk} \leftarrow (\mathsf{s}_1, \mathsf{s}_2)$
5: $\boldsymbol{pk} \leftarrow (\mathsf{a}, \mathsf{t})$

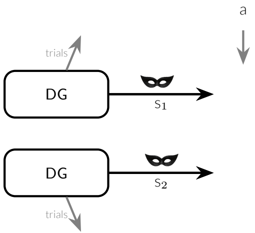


Masking GLP key generation

Algorithm 1 GLP key generation

Ensure: Signing key sk, verification key pk

1: $s_1, s_2 \xleftarrow{\$} \mathcal{R}_1$ //s_1 and s_2 have coefficients in $\{-1, 0, 1\}$
2: $a \xleftarrow{\$} \mathcal{R}$
3: $t \leftarrow as_1 + s_2$
4: $sk \leftarrow (s_1, s_2)$
5: $pk \leftarrow (a, t)$



Masking GLP key generation

Algorithm 1 GLP key generation

Ensure: Signing key sk, verification key pk

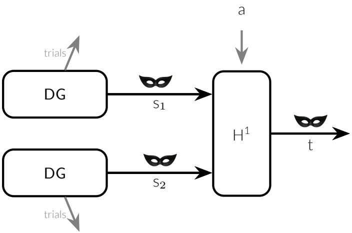
1: $s_1, s_2 \xleftarrow{\$} \mathcal{R}_1$ //s_1 and s_2 have coefficients in $\{-1, 0, 1\}$

2: $a \xleftarrow{\$} \mathcal{R}$

3: $t \leftarrow as_1 + s_2$

4: $sk \leftarrow (s_1, s_2)$

5: $pk \leftarrow (a, t)$



Masking GLP key generation

Algorithm 1 GLP key generation

Ensure: Signing key sk, verification key pk

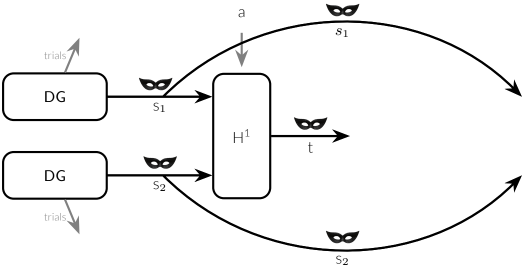
1: $\mathsf{s}_1, \mathsf{s}_2 \xleftarrow{\$} \mathcal{R}_1$ //s_1 and s_2 have coefficients in $\{-1, 0, 1\}$

2: $\mathsf{a} \xleftarrow{\$} \mathcal{R}$

3: $\mathsf{t} \leftarrow \mathsf{a}\mathsf{s}_1 + \mathsf{s}_2$

4: $sk \leftarrow (\mathsf{s}_1, \mathsf{s}_2)$

5: $pk \leftarrow (\mathsf{a}, \mathsf{t})$

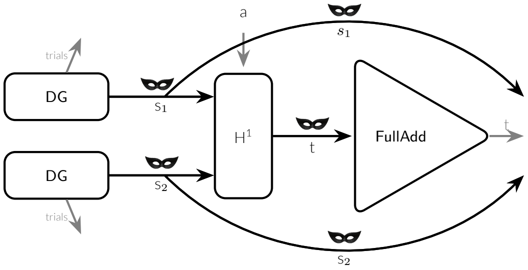


Masking GLP key generation

Algorithm 1 GLP key generation

Ensure: Signing key $\boldsymbol{sk}$, verification key $\boldsymbol{pk}$

1: $\mathsf{s}_1, \mathsf{s}_2 \xleftarrow{\$} \mathcal{R}_1$ //s_1 and s_2 have coefficients in $\{-1, 0, 1\}$
2: $\mathsf{a} \xleftarrow{\$} \mathcal{R}$
3: $\mathsf{t} \leftarrow \mathsf{a}\mathsf{s}_1 + \mathsf{s}_2$
4: $\boldsymbol{sk} \leftarrow (\mathsf{s}_1, \mathsf{s}_2)$
5: $\boldsymbol{pk} \leftarrow (\mathsf{a}, \mathsf{t})$

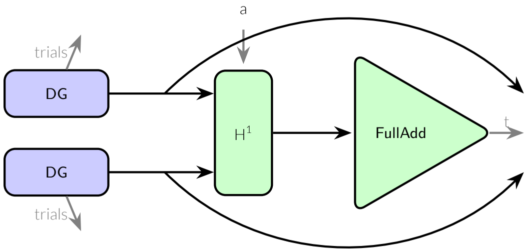


Masking GLP key generation

Algorithm 1 GLP key generation

Ensure: Signing key sk, verification key pk

1: $s_1, s_2 \xleftarrow{\$} \mathcal{R}_1$ //s_1 and s_2 have coefficients in $\{-1, 0, 1\}$
2: $a \xleftarrow{\$} \mathcal{R}$
3: $t \leftarrow as_1 + s_2$
4: $sk \leftarrow (s_1, s_2)$
5: $pk \leftarrow (a, t)$



Not masked

Non interferent

Non interferent with public output $trials$

Masking the signature

Algorithm 2 GLP sign

Require: $\mathsf{m}, pk = (\mathsf{a}, \mathsf{t}), sk = (\mathsf{s}_1, \mathsf{s}_2)$
Ensure: Signature σ

1: $\mathsf{y}_1, \mathsf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
2: $\mathsf{c} \leftarrow H(\mathsf{r} = \mathsf{a}\mathsf{y}_1 + \mathsf{y}_2, \mathsf{m})$
3: $\mathsf{z}_1 \leftarrow \mathsf{s}_1\mathsf{c} + \mathsf{y}_1$
4: $\mathsf{z}_2 \leftarrow \mathsf{s}_2\mathsf{c} + \mathsf{y}_2$
5: **if** z_1 or $\mathsf{z}_2 \notin \mathcal{R}_{k-\alpha}$ **then** restart
6: **return** $\sigma = (\mathsf{z}_1, \mathsf{z}_2, \mathsf{c})$

Masking the signature

Algorithm 2 GLP sign

Require: $m, pk = (a, t), sk = (s_1, s_2)$
Ensure: Signature σ
1: $y_1, y_2 \xleftarrow{\$} \mathcal{R}_k$
2: $c \leftarrow H(r = ay_1 + y_2, m)$
3: $z_1 \leftarrow s_1 c + y_1$
4: $z_2 \leftarrow s_2 c + y_2$
5: **if** z_1 or $z_2 \notin \mathcal{R}_{k-\alpha}$ **then** restart
6: **return** $\sigma = (z_1, z_2, c)$

Masking the commitment : unnecessary

Distinguishing (c, r) pairs from uniform is heuristically[1] a hard problem even for rejected executions.

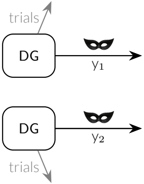
[1] Thanks' to V. Lyubashevsky, we also provided a non heuristic approach which requires somes changes in the algorithm

Masking the signature

Algorithm 3 Tweaked GLP sign

Require: $\mathrm{m}, \boldsymbol{pk} = (\mathrm{a}, \mathrm{t}), \boldsymbol{sk} = (\mathrm{s}_1, \mathrm{s}_2)$
Ensure: Signature $\boldsymbol{\sigma}$

$\mathrm{y}_1, \mathrm{y}_2 \xleftarrow{\$} \mathcal{R}_k$
$\mathrm{c} \leftarrow H(\mathrm{r} = \mathrm{a}\mathrm{y}_1 + \mathrm{y}_2, \mathrm{m})$
$\mathrm{z}_1 \leftarrow \mathrm{s}_1\mathrm{c} + \mathrm{y}_1$
$\mathrm{z}_2 \leftarrow \mathrm{s}_2\mathrm{c} + \mathrm{y}_2$
if z_1 or $\mathrm{z}_2 \notin \mathcal{R}_{k-\alpha}$ then return $\perp$
return $\boldsymbol{\sigma} = (\mathrm{z}_1, \mathrm{z}_2, \mathrm{c})$



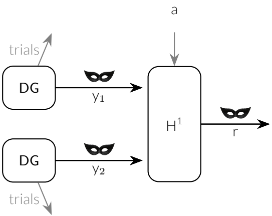
Masking the signature

Algorithm 3 Tweaked GLP sign

Require: $m, \boldsymbol{pk} = (a, t), \boldsymbol{sk} = (s_1, s_2)$

Ensure: Signature $\boldsymbol{\sigma}$

1: $y_1, y_2 \xleftarrow{\$} \mathcal{R}_k$
2: $c \leftarrow H(r = ay_1 + y_2, m)$
3: $z_1 \leftarrow s_1 c + y_1$
4: $z_2 \leftarrow s_2 c + y_2$
5: if z_1 or $z_2 \notin \mathcal{R}_{k-\alpha}$ then return $\perp$
6: return $\boldsymbol{\sigma} = (z_1, z_2, c)$



Masking the signature

Algorithm 3 Tweaked GLP sign

Require: $m, pk = (a, t), sk = (s_1, s_2)$

Ensure: Signature σ

1: $y_1, y_2 \xleftarrow{\$} \mathcal{R}_k$

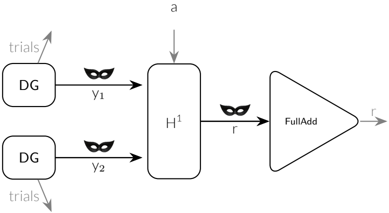
2: $c \leftarrow H(r = ay_1 + y_2, m)$

3: $z_1 \leftarrow s_1 c + y_1$

4: $z_2 \leftarrow s_2 c + y_2$

5: if z_1 or $z_2 \notin \mathcal{R}_{k-\alpha}$ then return $\perp$

6: return $\sigma = (z_1, z_2, c)$



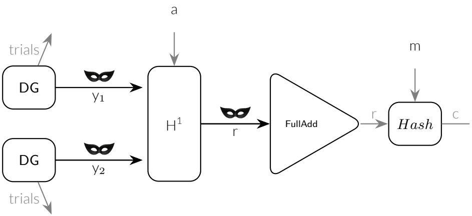
Masking the signature

Algorithm 3 Tweaked GLP sign

Require: $m, \boldsymbol{pk} = (a, t), \boldsymbol{sk} = (s_1, s_2)$

Ensure: Signature $\boldsymbol{\sigma}$

1: $y_1, y_2 \xleftarrow{\$} \mathcal{R}_k$
2: $c \leftarrow \boldsymbol{H}(r = ay_1 + y_2, m)$
3: $z_1 \leftarrow s_1 c + y_1$
4: $z_2 \leftarrow s_2 c + y_2$
5: if z_1 or $z_2 \notin \mathcal{R}_{k-\alpha}$ then return $\perp$
6: return $\boldsymbol{\sigma} = (z_1, z_2, c)$



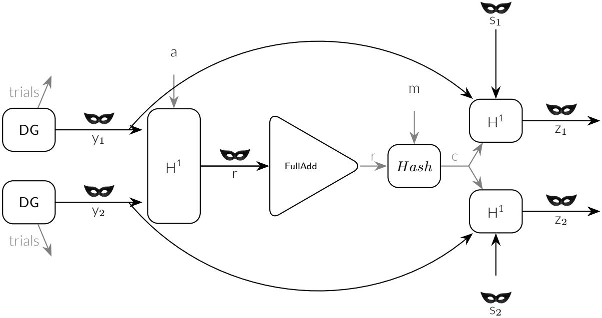
Masking the signature

Algorithm 3 Tweaked GLP sign

Require: $m, \boldsymbol{pk} = (a, t), \boldsymbol{sk} = (s_1, s_2)$

Ensure: Signature $\boldsymbol{\sigma}$

1: $y_1, y_2 \xleftarrow{\$} \mathcal{R}_k$
2: $c \leftarrow \boldsymbol{H}(r = ay_1 + y_2, m)$
3: $z_1 \leftarrow s_1 c + y_1$
4: $z_2 \leftarrow s_2 c + y_2$
5: if z_1 or $z_2 \notin \mathcal{R}_{k-\alpha}$ then return $\perp$
6: return $\boldsymbol{\sigma} = (z_1, z_2, c)$

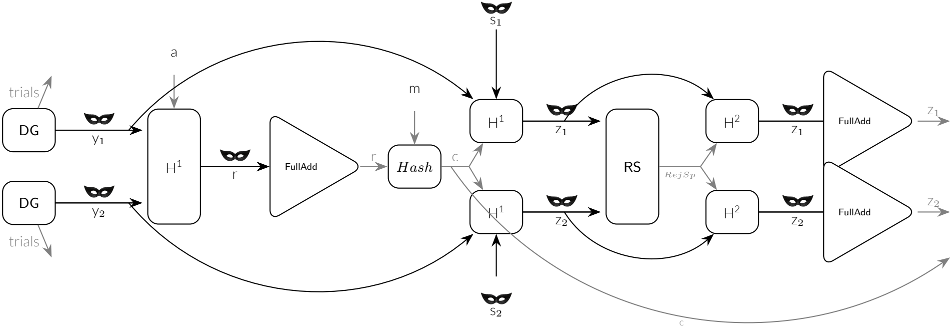


Masking the signature

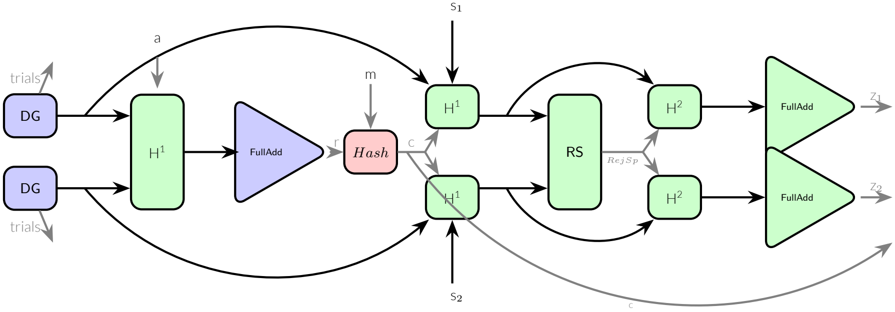
Algorithm 3 Tweaked GLP sign

Require: $m, \boldsymbol{pk} = (a, t), \boldsymbol{sk} = (s_1, s_2)$
Ensure: Signature $\boldsymbol{\sigma}$

1: $y_1, y_2 \xleftarrow{\$} \mathcal{R}_k$
2: $c \leftarrow \boldsymbol{H}(r = ay_1 + y_2, m)$
3: $z_1 \leftarrow s_1 c + y_1$
4: $z_2 \leftarrow s_2 c + y_2$
5: if z_1 or $z_2 \notin \mathcal{R}_{k-\alpha}$ then return $\perp$
6: return $\boldsymbol{\sigma} = (z_1, z_2, c)$



Composition



Not masked | Non interferent | Non interferent with public outputs $trials$ and r

Conversions Boolean to arithmetic

Proving the non interference of certain blocks (Rejection Sampling, Data Generation) was challenging

Algorithm 2 GLP signature

Require: $\mathsf{m}, \boldsymbol{pk}, \boldsymbol{sk}$
Ensure: Signature $\boldsymbol{\sigma}$

1: $\mathsf{y}_1, \mathsf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
2: $\mathsf{c} \leftarrow H(\mathsf{r} = \mathsf{a}\mathsf{y}_1 + \mathsf{y}_2, \mathsf{m})$
3: $\mathsf{z}_1 \leftarrow \mathsf{s}_1\mathsf{c} + \mathsf{y}_1$
4: $\mathsf{z}_2 \leftarrow \mathsf{s}_2\mathsf{c} + \mathsf{y}_2$
5: **if** z_1 or $\mathsf{z}_2 \notin \mathcal{R}_{k-\alpha}$ **then** restart
6: **return** $\boldsymbol{\sigma} = (\mathsf{z}_1, \mathsf{z}_2, \mathsf{c})$

$$\sum_{i=0}^{i=d} z_{1,i} \mod p \quad \leq \quad k - \alpha? \tag{1}$$

We had to adapt **arithmetic to Boolean conversions** from Coron, Großschädl and Vadnala in [CGV14].

$$\sum_{i=0}^{i=d} z_{1,i} \mod p \quad \rightarrow \quad \bigoplus_{i=0}^{i=d} z'_{1,i} \tag{2}$$

Performances

Table 1: Performances

Number of shares $(d+1)$	Unprotected	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{5}$	$\mathbf{6}$
Total CPU time (s)	0.540	8.15	16.4	39.5	62.1	111
Penalty factor	—	$\times 15$	$\times 30$	$\times 73$	$\times 115$	$\times 206$

Timings are provided for $\mathbf{100}$ executions of the signing algorithm, on one core of an Intel Core i7-3770 CPU-based desktop machine.

- The code will be published soon
- Quite promising in view of the lack of optimization

Future work

In a nutshell,

- Provable masked implementation of **GLP** signature scheme
- New security notions adapted to Fiat–Shamir framework.

➳ Can be applied directly to **Dilithium** (implementation in progress, Vincent Migliore)

BLISS and **Dilithium-G**

➳ Gaussians

➳ Not sure the Hash function can be unmasked

Conclusion

Thank you for your attention

Questions ?

Blog article on the RISQ project webpage : http://risq.fr/?page_id=365&lang=en

Eprint : https://eprint.iacr.org/2018/381

- Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala.
  Secure conversion between Boolean and arithmetic masking of any order.
  In Lejla Batina and Matthew Robshaw, editors, CHES 2014, volume 8731 of LNCS, pages 188–205. Springer, Heidelberg, September 2014.
- Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi.
  Towards sound approaches to counteract power-analysis attacks.
  In Michael J. Wiener, editor, CRYPTO'99, volume 1666 of LNCS, pages 398–412. Springer, Heidelberg, August 1999.
- Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann.
  Practical lattice-based cryptography: A signature scheme for embedded systems.
  In Emmanuel Prouff and Patrick Schaumont, editors, CHES 2012, volume 7428 of LNCS, pages 530–547. Springer, Heidelberg, September 2012.
- Yuval Ishai, Amit Sahai, and David Wagner.
  Private circuits: Securing hardware against probing attacks.
  In Dan Boneh, editor, CRYPTO 2003, volume 2729 of LNCS, pages 463–481. Springer, Heidelberg, August 2003.
- Vadim Lyubashevsky.
  Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures.
  In Mitsuru Matsui, editor, ASIACRYPT 2009, volume 5912 of LNCS, pages 598–616. Springer, Heidelberg, December 2009.

Vadim Lyubashevsky.
Lattice signatures without trapdoors.
In David Pointcheval and Thomas Johansson, editors, EUROCRYPT 2012, volume 7237 of LNCS, pages 738–755. Springer, Heidelberg, April 2012.

Emmanuel Prouff and Matthieu Rivain.
Masking against side-channel attacks: A formal security proof.
In Thomas Johansson and Phong Q. Nguyen, editors, EUROCRYPT 2013, volume 7881 of LNCS, pages 142–159. Springer, Heidelberg, May 2013.

Conversions Boolean to arithmetic

- **DG:** generation of sharings for coefficients $x \in [-k, k]$ $(k = 1)$

Conversions Boolean to arithmetic

- **DG:** generation of sharings for coefficients $x \in [-k, k]$ $(k = 1)$
  1. generate a Boolean sharing of x:

$$\forall 0 \leq i \leq d, \ x_i \leftarrow [0, 2^{w_0} - 1]$$

  where $2^{w_0} > 2k + 1 \geq 2^{w_0 - 1}$

Conversions Boolean to arithmetic

- **DG:** generation of sharings for coefficients $x \in [-k, k]$ $(k = 1)$
  1. generate a Boolean sharing of x:

     $$\forall 0 \leq i \leq d,\ \ x_i \leftarrow [0, 2^{w_0} - 1]$$

     where $2^{w_0} > 2k + 1 \geq 2^{w_0 - 1}$
  2. $(\delta_i)_{0 \leq i \leq d} \leftarrow (\mathsf{x}_i)_{0 \leq i \leq d} - (\mathsf{k}_i)_{0 \leq i \leq d}$

Conversions Boolean to arithmetic

- **DG:** generation of sharings for coefficients $x \in [-k, k]$ $(k = 1)$
  1. generate a Boolean sharing of x:

     $$\forall 0 \leq i \leq d,\ \ x_i \leftarrow [0, 2^{w_0} - 1]$$

     where $2^{w_0} > 2k + 1 \geq 2^{w_0 - 1}$
  2. $(\delta_i)_{0 \leq i \leq d} \leftarrow (\mathsf{x}_i)_{0 \leq i \leq d} - (\mathsf{k}_i)_{0 \leq i \leq d}$
  3. $b \leftarrow$ unmask δ's most significant bit
  4. b equals 0 iff $x \geq 2k + 1$

Conversions Boolean to arithmetic

- **DG:** generation of sharings for coefficients $x \in [-k, k]$ $(k = 1)$
  1. generate a Boolean sharing of x:
     $$\forall 0 \leq i \leq d,\ \ x_i \leftarrow [0, 2^{w_0} - 1]$$
     where $2^{w_0} > 2k + 1 \geq 2^{w_0 - 1}$
  2. $(\delta_i)_{0 \leq i \leq d} \leftarrow (\mathsf{x}_i)_{0 \leq i \leq d} - (\mathsf{k}_i)_{0 \leq i \leq d}$
  3. $b \leftarrow$ unmask δ's most significant bit
  4. b equals 0 iff $x \geq 2k + 1$
  5. convert $(\mathsf{x}_i)_{0 \leq i \leq d}$ to an arithmetic masking

Conversions Boolean to arithmetic

- **DG:** generation of sharings for coefficients $x \in [-k, k]$ $(k = 1)$
  1. generate a Boolean sharing of x:
  $$\forall 0 \leq i \leq d, \; x_i \leftarrow [0, 2^{w_0} - 1]$$
  where $2^{w_0} > 2k + 1 \geq 2^{w_0 - 1}$
  2. $(\delta_i)_{0 \leq i \leq d} \leftarrow (\mathsf{x}_i)_{0 \leq i \leq d} - (\mathsf{k}_i)_{0 \leq i \leq d}$
  3. $b \leftarrow$ unmask δ's most significant bit
  4. b equals 0 iff $x \geq 2k + 1$
  5. convert $(\mathsf{x}_i)_{0 \leq i \leq d}$ to an arithmetic masking
- **Rejection Sampling:** are coefficients of z_1 in $[-k + \alpha, k - \alpha]$?

Conversions Boolean to arithmetic

- **DG:** generation of sharings for coefficients $x \in [-k, k]$ $(k = 1)$
  1. generate a Boolean sharing of x:
  $$\forall 0 \leq i \leq d, \;\; x_i \leftarrow [0, 2^{w_0} - 1]$$
  where $2^{w_0} > 2k + 1 \geq 2^{w_0 - 1}$
  2. $(\delta_i)_{0 \leq i \leq d} \leftarrow (\mathsf{x}_i)_{0 \leq i \leq d} - (\mathsf{k}_i)_{0 \leq i \leq d}$
  3. $b \leftarrow$ unmask δ's most significant bit
  4. b equals 0 iff $x \geq 2k + 1$
  5. convert $(\mathsf{x}_i)_{0 \leq i \leq d}$ to an arithmetic masking
- **Rejection Sampling:** are coefficients of z_1 in $[-k + \alpha, k - \alpha]$?
  1. convert mod-p arithmetic sharing into Boolean masking

Conversions Boolean to arithmetic

- **DG:** generation of sharings for coefficients $x \in [-k, k]$ $(k = 1)$
  1. generate a Boolean sharing of x:
  $$\forall 0 \leq i \leq d, \;\; x_i \leftarrow [0, 2^{w_0} - 1]$$
  where $2^{w_0} > 2k + 1 \geq 2^{w_0 - 1}$
  2. $(\delta_i)_{0 \leq i \leq d} \leftarrow (\mathsf{x}_i)_{0 \leq i \leq d} - (\mathsf{k}_i)_{0 \leq i \leq d}$
  3. $b \leftarrow$ unmask δ's most significant bit
  4. b equals 0 iff $x \geq 2k + 1$
  5. convert $(\mathsf{x}_i)_{0 \leq i \leq d}$ to an arithmetic masking
- **Rejection Sampling:** are coefficients of z_1 in $[-k + \alpha, k - \alpha]$?
  1. convert mod-p arithmetic sharing into Boolean masking
  2. as in Data Generation, compute the masked difference with $k - \alpha$ difference

Conversions Boolean to arithmetic

- **DG:** generation of sharings for coefficients $x \in [-k, k]$ $(k = 1)$
  1. generate a Boolean sharing of x:
  $$\forall 0 \leq i \leq d, \;\; x_i \leftarrow [0, 2^{w_0} - 1]$$
  where $2^{w_0} > 2k + 1 \geq 2^{w_0 - 1}$
  2. $(\delta_i)_{0 \leq i \leq d} \leftarrow (\mathsf{x}_i)_{0 \leq i \leq d} - (\mathsf{k}_i)_{0 \leq i \leq d}$
  3. $b \leftarrow$ unmask δ's most significant bit
  4. b equals 0 iff $x \geq 2k + 1$
  5. convert $(\mathsf{x}_i)_{0 \leq i \leq d}$ to an arithmetic masking
- **Rejection Sampling:** are coefficients of z_1 in $[-k + \alpha, k - \alpha]$?
  1. convert mod-p arithmetic sharing into Boolean masking
  2. as in Data Generation, compute the masked difference with $k - \alpha$ difference
  3. securely check the most significant bit