An Efficient and Provable Masked Implementation of qTESLA

François Gérard and Mélissa Rossi

Post-quantum

Most of currently used public-key algorithms are vulnerable to quantum cryptanalysis.

Post-quantum

Most of currently used public-key algorithms are vulnerable to quantum cryptanalysis. Sure, quantum computers are not there yet but...

Post-quantum

Most of currently used public-key algorithms are vulnerable to quantum cryptanalysis. Sure, quantum computers are not there yet but...

Things that take time:

- Building secure and practical schemes
- Getting confidence in the underlying assumption
- Deploying the scheme outside of academia

Post-quantum

Most of currently used public-key algorithms are vulnerable to quantum cryptanalysis. Sure, quantum computers are not there yet but...

Things that take time:

- Building secure and practical schemes
- Getting confidence in the underlying assumption
- Deploying the scheme outside of academia

Risks are too high and post-quantum security *might* be needed right now

Practical aspects

NIST post-quantum standardization project started in 2017, its first round ended in early 2019

> *Performances will play a larger role in round 2*
>
> *NIST (basically)*

Practical aspects

NIST post-quantum standardization project started in 2017, its first round ended in early 2019

> *Performances will play a larger role in round 2*
>
> *NIST (basically)*

Performances are mostly critical on embedded devices:

- Need for efficient implementations (libpqm4)
- Need for side-channel countermeasures

Masking

Values are split into $N + 1$ shares such that any set of N shares does not reveal anything about the masked value

Masking

Values are split into $N+1$ shares such that any set of N shares does not reveal anything about the masked value

$$v = \bigoplus v_i$$	$$v = \sum v_i$$
Boolean masking	Arithmetic masking

Masking

Values are split into $N + 1$ shares such that any set of N shares does not reveal anything about the masked value

$v = \bigoplus v_i$	$v = \sum v_i$
Boolean masking	Arithmetic masking

In the following, a value v split in $N + 1$ shares will be written $(v_i)_{0 \leq i \leq N}$ or (v_i) for short

qTESLA

- Lattice-based Schnorr-like signature candidate in the NIST project

qTESLA

- Lattice-based Schnorr-like signature candidate in the NIST project
- Security assumption (Ring Learning With Errors) : In $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$, it is hard to find $\mathbf{s}$ (or $\mathbf{e}$) from $\mathbf{t} \leftarrow \mathbf{a} \cdot \mathbf{s} + \mathbf{e}$

qTESLA

- Lattice-based Schnorr-like signature candidate in the NIST project
- Security assumption (Ring Learning With Errors) : In $\mathbb{Z}_q[X]/\langle X^n + 1 \rangle$, it is hard to find $\mathbf{s}$ (or $\mathbf{e}$) from $\mathbf{t} \leftarrow \mathbf{a} \cdot \mathbf{s} + \mathbf{e}$

Parameters	`qTESLA-I`	`qTESLA-III`	Description
n	512	1024	Dimension of the ring
q	$\approx 2^{22}$	$\approx 2^{23}$	Modulus
E	1586	1147	Rejection parameter
S	1586	1233	Rejection parameter
B	$2^{20} - 1$	$2^{21} - 1$	Bound for $\mathbf{y}$
d	21	22	Bits dropped in $[\cdot]_M$

qTESLA

Disclaimer

The practical results of this work are based on the heuristic parameter sets of `qTESLA` that where removed during the review phase of this conference. Our masking scheme still applies but the code has to be changed to match the submission.

State of the art

Previously:

- Masking of GLP + code + proofs (Eurocrypt 2018)
- Masking of Dilithium + code + experiments (ACNS 2019)

State of the art

Previously:

- Masking of GLP + code + proofs (Eurocrypt 2018)
- Masking of Dilithium + code + experiments (ACNS 2019)

Our work:

- Masking of `qTESLA`
- Optimization for order 1
- Proofs in the ISW model
- Public implementation in the code of the submission

Idea of the scheme

Public parameter: $\mathbf{a}$
Secret key: $\mathbf{s}, \mathbf{e}$
Public key: $\mathbf{t} \leftarrow \mathbf{a} \cdot \mathbf{s} + \mathbf{e}$

Sign$(\mathbf{s}, m)$:

1: **do**
2: $\quad \mathbf{y} \xleftarrow{r} Y$
3: $\quad \mathbf{c} \leftarrow H(\lfloor \mathbf{a} \cdot \mathbf{y} \rceil, m)$
4: $\quad \mathbf{z} \leftarrow \mathbf{s} \cdot \mathbf{c} + \mathbf{y}$
5: **while** `Rejected`$(\mathbf{z})$
6: **and not** `WellRounded`$(\mathbf{a} \cdot \mathbf{y})$
7: **return** $\mathbf{z}, \mathbf{c}$

Verify$(\mathbf{z}, \mathbf{c}, \mathbf{t}, m)$:

1: $\mathbf{v} \leftarrow \mathbf{a} \cdot \mathbf{z} - \mathbf{t} \cdot \mathbf{c} = \mathbf{a} \cdot \mathbf{y} - \mathbf{e} \cdot \mathbf{c}$
2: **return** 1 **if** $\mathbf{c} = H(\lfloor \mathbf{v} \rceil_M, m)$ **and** $\mathbf{z}$ **is small else** 0

qTESLA sign$(\mathbf{s}, m)$

$\text{counter} \leftarrow 1$
$r \xleftarrow{r} \{0,1\}^{\kappa}$
$\text{rand} \leftarrow \mathsf{PRF}(\text{seed}_y, r, \mathsf{H}(m))$
$\mathbf{y} \leftarrow \mathsf{ySampler}(\text{rand, counter})$
$\mathbf{a} \leftarrow \mathsf{GenA}(\text{seed}_a)$
$\mathbf{v} \leftarrow \mathbf{a} \cdot \mathbf{y} \bmod^{\pm} q$
$\mathbf{c} \leftarrow \mathsf{Enc}(\mathsf{H}([\mathbf{v}]_M, m))$
$\mathbf{z} \leftarrow \mathbf{y} + \mathbf{s} \cdot \mathbf{c}$
if $\mathbf{z} \notin \mathcal{R}_{q,[B-S]}$ **then**
  $\text{counter} \leftarrow \text{counter} + 1$
  goto 4
end if
$\mathbf{w} \leftarrow \mathbf{v} - \mathbf{e} \cdot \mathbf{c} \bmod^{\pm} q$
if $||[\mathbf{w}]_L||_\infty \geq 2^{d-1} - E$
 or $||\mathbf{w}||_\infty \geq \lfloor q/2 \rfloor - E$ **then**
  $\text{counter} \leftarrow \text{counter} + 1$
  goto 4
end if
return $(\mathbf{z}, \mathbf{c})$

Sensitive parts

counter $\leftarrow 1$
$r \xleftarrow{r} \{0,1\}^{\kappa}$
rand $\leftarrow \mathsf{PRF}(\mathrm{seed}_y, r, \mathsf{H}(m))$
$\mathbf{y} \leftarrow \mathsf{ySampler}(\mathrm{rand},\ \mathrm{counter})$
$\mathbf{a} \leftarrow \mathsf{GenA}(\mathrm{seed}_a)$
$\mathbf{v} \leftarrow \mathbf{a} \cdot \mathbf{y} \bmod^{\pm} q$
$\mathbf{c} \leftarrow \mathsf{Enc}(\mathsf{H}([\mathbf{v}]_M, m))$
$\mathbf{z} \leftarrow \mathbf{y} + \mathbf{s} \cdot \mathbf{c}$

if $\mathbf{z} \notin \mathcal{R}_{q,[B-S]}$ **then**
    counter $\leftarrow$ counter $+ 1$
    goto 4
end if
$\mathbf{w} \leftarrow \mathbf{v} - \mathbf{e} \cdot \mathbf{c} \bmod^{\pm} q$
if $\|[\mathbf{w}]_L\|_\infty \geq 2^{d-1} - E$
 or $\|\mathbf{w}\|_\infty \geq \lfloor q/2 \rfloor - E$ **then**
    counter $\leftarrow$ counter $+ 1$
    goto 4
end if
return $(\mathbf{z}, \mathbf{c})$

Modifications - PRF removal

- If two different signatures use the same $\mathbf{y}$, the secret key is trivially revealed

Modifications - PRF removal

- If two different signatures use the same $\mathbf{y}$, the secret key is trivially revealed
- Goal of the PRF is to avoid nonce reuse under the collision resistance assumption

Modifications - PRF removal

- If two different signatures use the same $\mathbf{y}$, the secret key is trivially revealed
- Goal of the PRF is to avoid nonce reuse under the collision resistance assumption
- Nevertheless security is only based on the randomness of $\mathbf{y}$

Modifications - PRF removal

- If two different signatures use the same $\mathbf{y}$, the secret key is trivially revealed
- Goal of the PRF is to avoid nonce reuse under the collision resistance assumption
- Nevertheless security is only based on the randomness of $\mathbf{y}$
- Since masking the PRF would be a significant overhead and using a masking scheme is assuming having access to a reasonable RNG, we removed the PRF.

Modifications - Power of two modulus

- `qTESLA` uses a prime q to instantiate its ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ to enable NTT-based algorithms for polynomial multiplication

Modifications - Power of two modulus

- `qTESLA` uses a prime q to instantiate its ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ to enable NTT-based algorithms for polynomial multiplication
- As pointed out in previous works, masked modular arithmetic is very expensive

Modifications - Power of two modulus

- `qTESLA` uses a prime q to instantiate its ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ to enable NTT-based algorithms for polynomial multiplication
- As pointed out in previous works, masked modular arithmetic is very expensive
- One solution is to use a power of two modulus as reduction is a mask on shares

Modifications - Power of two modulus

- `qTESLA` uses a prime q to instantiate its ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ to enable NTT-based algorithms for polynomial multiplication
- As pointed out in previous works, masked modular arithmetic is very expensive
- One solution is to use a power of two modulus as reduction is a mask on shares
- Polynomial multiplication is slower (Karatsuba) ... but is not the bottleneck any more in a masked setting

Main components to mask

ySampler $\rightarrow$ already state of the art

PolynomialMul $\rightarrow$ not needed since $K \cdot \sum_i s_i = \sum_i K \cdot s_i$

RejectionSampling $\rightarrow x \in [-B, \ldots, B]$

Rounding $\rightarrow (w \bmod^{\pm} q - [w]_L)/2^d$

WellRounded $\rightarrow |x| < \lfloor q/2 \rfloor - E \;\; \textbf{and} \;\; |[x]_L| < 2^{d-1}$

Toolbox of gadgets

$\mathsf{SecAnd}((a_i),(b_i)) = (c_i)$ s.t. $\bigoplus_i c_i = (\bigoplus_i a_i)\&(\bigoplus_i b_i)$

Toolbox of gadgets

$\mathsf{SecAnd}((a_i),(b_i)) = (c_i)$ s.t. $\bigoplus_i c_i = (\bigoplus_i a_i) \& (\bigoplus_i b_i)$

$\mathsf{SecAdd}((a_i),(b_i)) = (c_i)$ s.t. $\bigoplus_i c_i = (\bigoplus_i a_i) + (\bigoplus_i b_i)$

Toolbox of gadgets

$\mathsf{SecAnd}((a_i),(b_i)) = (c_i)$ s.t. $\bigoplus_i c_i = (\bigoplus_i a_i)\&(\bigoplus_i b_i)$

$\mathsf{SecAdd}((a_i),(b_i)) = (c_i)$ s.t. $\bigoplus_i c_i = (\bigoplus_i a_i) + (\bigoplus_i b_i)$

$\mathsf{SecArithBoolModq}((a_i)) = (a'_i)$ s.t $(\bigoplus_i a'_i) = (\sum_i a_i)\%q$

Toolbox of gadgets

$\mathsf{SecAnd}((a_i),(b_i)) = (c_i)$ s.t. $\bigoplus_i c_i = (\bigoplus_i a_i)\&(\bigoplus_i b_i)$

$\mathsf{SecAdd}((a_i),(b_i)) = (c_i)$ s.t. $\bigoplus_i c_i = (\bigoplus_i a_i) + (\bigoplus_i b_i)$

$\mathsf{SecArithBoolModq}((a_i)) = (a'_i)$ s.t $(\bigoplus_i a'_i) = (\sum_i a_i)\%q$

$\mathsf{FullXor}((a_i)) = \bigoplus a_i$

Masked Absolute Value

Use the good ol' trick:

- $m \leftarrow x >> 31$
- $|x| \leftarrow (x + m) \oplus m$

Masked Absolute Value

Use the good ol' trick:

- $m \leftarrow x >> 31$
- $|x| \leftarrow (x + m) \oplus m$

1: $(mask_i)_{0 \leq i \leq N} \leftarrow ((x_i)_{0 \leq i \leq N} << (\mathrm{RADIX} - k)) >> (\mathrm{RADIX} - 1))$
2: $(x'_i)_{0 \leq i \leq N} \leftarrow \mathsf{Refresh}((x_i)_{0 \leq i \leq N})$
3: $(x_i)_{0 \leq i \leq N} \leftarrow \mathsf{SecAdd}((x'_i)_{0 \leq i \leq N}, (mask_i)_{0 \leq i \leq N}))$
4: $(|x|_i)_{0 \leq i \leq N} \leftarrow ((x_i)_{0 \leq i \leq N} \oplus (mask_i)_{0 \leq i \leq N}) \wedge (2^k - 1)$

Masked rejection sampling

Compare with subtract and shift:

- $t \leftarrow |x| - (\mathrm{BOUND}{+}1)$
- $b \leftarrow t >> 31$

Masked rejection sampling

Compare with subtract and shift:

- $t \leftarrow |x| - (\text{BOUND}{+}1)$
- $b \leftarrow t >> 31$

1: $(\text{SUP}_i)_{0\leq i\leq N} \leftarrow (-B + S - 1, 0, ..., 0)$
2: $(a'_i)_{0\leq i\leq N} \leftarrow \mathsf{GenSecArithBoolModq}((a_i)_{0\leq i\leq N})$
3: $(x_i)_{0\leq i\leq N} \leftarrow \mathsf{AbsVal}((a'_i)_{0\leq i\leq N}, \log_2 q)$
4: $(x_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((x_i)_{0\leq i\leq N}, (\text{SUP}_i)_{0\leq i\leq N})$
5: $(b_i)_{0\leq i\leq N} \leftarrow ((x_i)_{0\leq i\leq N} >> \text{RADIX} - 1)$
6: **return** $rs := \mathsf{FullXor}((b_i)_{0\leq i\leq N})$

Masked rounding

$$[\cdot]_L : \mathbb{Z} \to \mathbb{Z}, w \mapsto w \bmod^{\pm} 2^d$$
$$[\cdot]_M : \mathbb{Z} \to \mathbb{Z}, w \mapsto (w \bmod^{\pm} q - [w]_L)/2^d$$

where $x \bmod^{\pm} q$ denotes the unique integer $x_{ct} \in (-q/2, \ldots, q/2]$ such that $x_{ct} \equiv x \pmod{q}$

Masked rounding

$$[\cdot]_L : \mathbb{Z} \to \mathbb{Z}, w \mapsto w \bmod^{\pm} 2^d$$
$$[\cdot]_M : \mathbb{Z} \to \mathbb{Z}, w \mapsto (w \bmod^{\pm} q - [w]_L)/2^d$$

where $x \bmod^{\pm} q$ denotes the unique integer $x_{ct} \in (-q/2, \ldots, q/2]$ such that $x_{ct} \equiv x \pmod q$

$$\mathbb{Z}/8\mathbb{Z} = \{-3, -2, -1, 0, 1, 2, 3, 4\}$$

Masked rounding

- Compute $w \bmod^{\pm} q$:
  - $w = w \,\texttt{\%}\, q$
  - **if** $(w > q/2)$ **then** $w \mathrel{-}= q$

Masked rounding

- Compute $w \bmod^{\pm} q$:
  - $w = w \,\texttt{\%}\, q$
  - **if** $(w > q/2)$ **then** $w \mathrel{-}= q$
- Subtract $[w]_L$ and divide by 2^d:
  - $w \mathrel{+}= 2^{d-1} - 1$
  - $w >>= d$

Masked rounding

- Compute $w \bmod^{\pm} q$:
  - $w = w \,\texttt{\%}\, q$
  - **if** $(w > q/2)$ **then** $w \mathrel{-}= q$

- Subtract $[w]_L$ and divide by 2^d:
  - $w \mathrel{+}= 2^{d-1} - 1$
  - $w \mathrel{>>}= d$

Second part analogous to computing $\lceil x \rfloor$ as $\lfloor x + 0.4999\ldots \rfloor$

Masked Rounding

$(\mathrm{MINUS\_Q\_HALF}_i)_{0\le i\le N} \leftarrow (-q/2-1, 0, ..., 0)$

$(\mathrm{CONST}_i)_{0\le i\le N} \leftarrow (2^{d-1}-1, 0, ..., 0)$

/* w = w % q */

$(a'_i)_{0\le i\le N} \leftarrow \mathsf{GenSecArithBoolModq}(a_i)_{0\le i\le N}$

/* if (w > q/2) then w −= q */

$(b_i)_{0\le i\le N} \leftarrow \mathsf{SecAdd}((a'_i)_{0\le i\le N}, (\mathrm{MINUS\_Q\_HALF}_i)_{0\le i\le N})$

$b_0 = \neg b_0$

$(b_i)_{0\le i\le N} \leftarrow ((b_i)_{0\le i\le N} >> \mathrm{RADIX}-1) << \log_2 q$

$(a'_i)_{0\le i\le N} \leftarrow (a'_i)_{0\le i\le N} \oplus (b_i)_{0\le i\le N}$

/* w += $\mathbf{2^{d-1}-1}$ */

$(a'_i)_{0\le i\le N} \leftarrow \mathsf{SecAdd}((a'_i)_{0\le i\le N}, (\mathrm{CONST}_i)_{0\le i\le N})$

/*w >>= d */

$(a'_i)_{0\le i\le N} \leftarrow (a'_i)_{0\le i\le N} >> d$

return $u \coloneqq \mathsf{FullXor}((a'_i)_{0\le i\le N})$

Cycle count of individual gadgets

Masking order	Order 1	Order 2	Order 3	Order 4	Order 5
RG	98	410	840	1 328	2 416
MaskedRound	164	1 400	2 454	4 314	6 142
MaskedWR	280	2 080	3 914	6 432	9 034
MaskedRS	178	1 440	2 496	4 432	6 254
SecAdd	44	294	592	870	1 192
SecAnd	20	28	44	70	96
GenSecArith-BoolModQ	96	786	1 152	3 148	3 500
SecBoolArith	20	42	108	288	884

Fully masked signature

Masking order	Unmasked	Order 1	Order 2	Order 3	Order 4	Order 5
`qTESLA-I` (RNG off)	645 673	2 394 085	7 000 117	9 219 826	16 577 823	24 375 359
`qTESLA-I` (RNG on)	671 169	2 504 204	13 878 830	24 582 943	39 967 191	59 551 027
`qTESLA-I` (RNG on) Scaling	1	$\times 4$	$\times 21$	$\times 37$	$\times 60$	$\times 89$
`qTESLA-I` CortexM4	11 304 025	23 519 583	-	-	-	

Cycle count on Intel i7 laptop and ARM Cortex-M4.

RNG off means `rand_uint32()` always returns 0.

Number of calls to `rand_uint32()`

Masking order	Order 1	Order 2	Order 3	Order 4	Order 5
`qTESLA-I`	85 810	1 383 459	2 761 525	4 923 709	7 638 422
`qTESLA-III`	115 392	1 826 545	3 721 800	6 482 130	10 005 714

Order 2 masking already needs over 4MB of randomness !

Conclusion

- Overhead is huge for more than 2 shares

Conclusion

- Overhead is huge for more than 2 shares
- Simpler rounding would make masking easier

Conclusion

- Overhead is huge for more than 2 shares
- Simpler rounding would make masking easier
- Power of two modulus seems to help a lot

Conclusion

- Overhead is huge for more than 2 shares
- Simpler rounding would make masking easier
- Power of two modulus seems to help a lot
- Computational overhead mainly due to randomness generation

Conclusion

- Overhead is huge for more than 2 shares
- Simpler rounding would make masking easier
- Power of two modulus seems to help a lot
- Computational overhead mainly due to randomness generation
- Design of the signature could be improved (for masking) but lattices are quite masking friendly

?