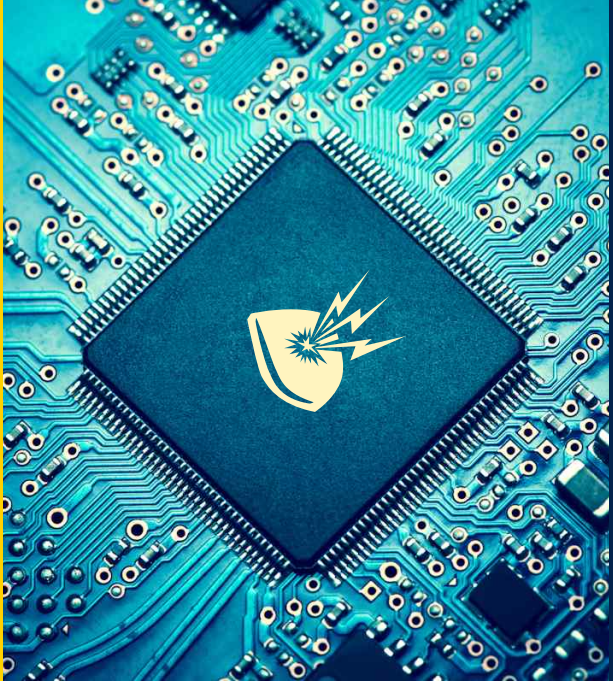


life.augmented

Masked Compression for ML-KEM

Guilhèm Assael[1], Gilles Van Assche[2]

[1] STMicroelectronics, Rousset, France
[2] STMicroelectronics, Diegem, Belgium

April 1st, 2026

Outline

ML-KEM ciphertext compression

Module Lattice Key-Encapsulation Mechanism (ML-KEM)

- **Post-quantum** Key-Encapsulation Mechanism.
- Standardized by NIST in 2024: **FIPS 203**.
- Relies on the hardness of **Learning With Errors** (LWE) over module lattices.
- **Arithmetic on polynomials** of $\mathbb{F}_q[X]/(X^{256}+1)$, where $q = 3329$ (12 bits).

ML-KEM ciphertext compression

Module Lattice Key-Encapsulation Mechanism (ML-KEM)

- **Post-quantum** Key-Encapsulation Mechanism.
- Standardized by NIST in 2024: **FIPS 203**.
- Relies on the hardness of **Learning With Errors** (LWE) over module lattices.
- **Arithmetic on polynomials** of $\mathbb{F}_q[X]/(X^{256}+1)$, where $q = 3329$ (12 bits).

Ciphertext compression ≈ rounded integer rescaling to d bits

Input coefficients $x \in \mathbb{F}_q = [\![0, q-1]\!]$ → $z = \mathrm{compress}_d(x) = \left\lfloor \frac{x 2^d}{q} \right\rceil$ → Compressed coefs $z \in [\![0, 2^d - 1]\!]$

Compression reduces the size of ML-KEM ciphertexts by 20–30 %.

Masking against side-channel attacks

Masking counters side-channel attacks by splitting sensitive values into $n \geqslant 2$ **shares**.

- Each of the n shares is independent from the secret
- Recombining the shares gives back the secret $x_0 \star x_1 \star \cdots \star x_{n-1} = x$
- $\star$ is a group operation, e.g. arithmetic sum mod q or mod 2^d, bit-wise exclusive-or

A **gadget** is the masked implementation of a given function.

Masking against side-channel attacks

Masking counters side-channel attacks by splitting sensitive values into $n \geqslant 2$ **shares**.

- Each of the n shares is independent from the secret
- Recombining the shares gives back the secret $x_0 \star x_1 \star \cdots \star x_{n-1} = x$
- $\star$ is a group operation, e.g. arithmetic sum mod q or mod 2^d, bit-wise exclusive-or

A **gadget** is the masked implementation of a given function.

The t-probing model – PINI composability

The t-probing model considers that attackers can place up to t probes on wires of a gadget, revealing its internal values.
Probe-Isolating Non-Interfering (PINI) gadgets are secure against $n - 1$ probes.
PINI gadgets can be trivially and securely **composed** into larger PINI gadgets.

ML-KEM compression must be protected

1-bit compression for message decoding ($d = 1$)

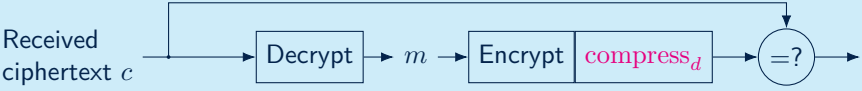
Noisy decrypted coefficient $c = m\frac{q+1}{2} + \epsilon \in [\![0, q-1]\!]$ → $\mathrm{compress}_1$ → 1 bit of message $m \in \{0, 1\}$

ML-KEM compression must be protected

1-bit compression for message decoding ($d = 1$)

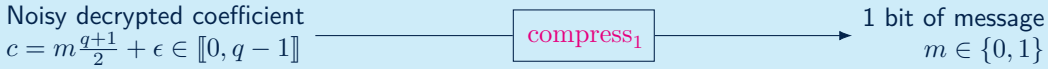
Noisy decrypted coefficient $c = m\frac{q+1}{2} + \epsilon \in [\![0, q-1]\!]$ → compress$_1$ → 1 bit of message $m \in \{0, 1\}$

d-bit compression during reencryption ($d \in \{4, 5, 10, 11\}$)



ML-KEM compression must be protected

1-bit compression for message decoding ($d = 1$)



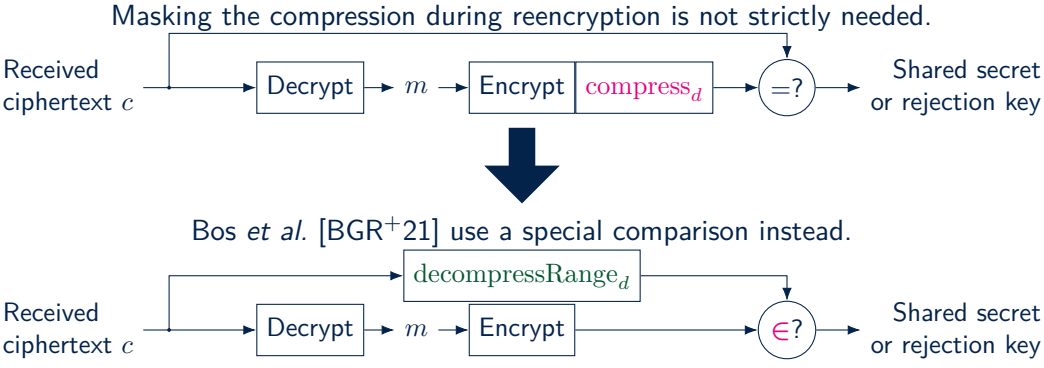
d-bit compression during reencryption ($d \in \{4, 5, 10, 11\}$)

Both operations process sensitive data

- The decrypted message (before/after decoding) is sensitive: message-recovery attack
- The re-encrypted ciphertext is sensitive: decryption-failure oracle

Workaround to masking the compression

Bos *et al.*'s decompressed comparison [BGR+21]



This solution only works because we don't *actually* need the compressed result.

High-order masked compression

Coron *et al.*'s [CGMZ23] extension of [FBR$^+$22]

$$z = \mathrm{compress}_d\left(\sum_{i=0}^{n-1} x_i\right) = \left\lfloor \frac{(\sum_{i=0}^{n-1} x_i)2^d}{q} \right\rceil \bmod 2^d \approx \left\lfloor \sum_{i=0}^{n-1} \mathrm{round}_\alpha\left(\frac{x_i 2^d}{q}\right) \right\rceil \bmod 2^d$$

round_α: Round to fixed-point value with α fractional bits

With $\alpha = \lceil \log_2(nq) \rceil$, rounding errors sum to $\epsilon \leqslant n2^{-\alpha} < 1/q$ and the final result is correct.

[FBR$^+$22] Fritzmann *et al.* Masked accelerators and instruction set extensions for PQC. TCHES 2022/1.
[CGMZ23] Coron *et al.* High-order polynomial comparison and masking lattice-based encryption. TCHES 2023/1.

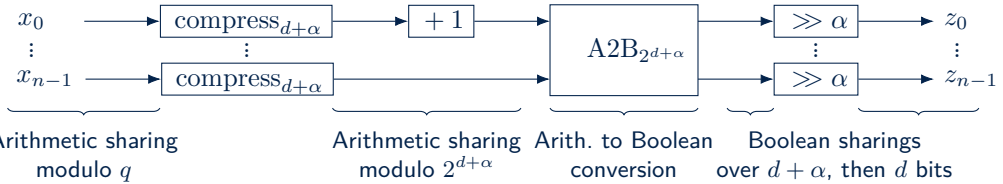
High-order masked compression

Coron *et al.*'s [CGMZ23] extension of [FBR$^+$22]

$$z = \mathrm{compress}_d\left(\sum_{i=0}^{n-1} x_i\right) = \left\lfloor \frac{(\sum_{i=0}^{n-1} x_i)2^d}{q} \right\rceil \bmod 2^d \approx \left\lfloor \sum_{i=0}^{n-1} \mathrm{round}_\alpha\left(\frac{x_i 2^d}{q}\right) \right\rceil \bmod 2^d$$

round_α: Round to fixed-point value with α fractional bits

With $\alpha = \lceil \log_2(nq) \rceil$, rounding errors sum to $\epsilon \leqslant n2^{-\alpha} < 1/q$ and the final result is correct.



[FBR$^+$22] Fritzmann *et al.* Masked accelerators and instruction set extensions for PQC. TCHES 2022/1.
[CGMZ23] Coron *et al.* High-order polynomial comparison and masking lattice-based encryption. TCHES 2023/1.

Managing the compression error better

Fritzmann *et al.* and Coron *et al.* *reduce* the error to a *negligible* level.

We **keep track** of the compression error and **correct it**.

Managing the compression error better

Fritzmann *et al.* and Coron *et al.* *reduce* the error to a *negligible* level.

We **keep track** of the compression error and **correct it**.

Compress losslessly by storing the **exact** quantization error for each share x_i.

$$\left.\begin{matrix} y_i \\ f_i \end{matrix}\right\} = \mathrm{DivMod}_d(x) = \left\{\begin{matrix} \lfloor (2^d x_i + r_i) & / & q \rfloor \bmod 2^d \\ (2^d x_i + r_i) & \bmod & q \end{matrix}\right. \quad \text{with } \sum r_i = \tfrac{q+1}{2}$$

Managing the compression error better

Fritzmann *et al.* and Coron *et al.* *reduce* the error to a *negligible* level.

We **keep track** of the compression error and **correct it**.

Compress losslessly by storing the **exact** quantization error for each share x_i.

$$\left.\begin{matrix} y_i \\ f_i \end{matrix}\right\} = \mathrm{DivMod}_d(x) = \left\{\begin{matrix} \lfloor (2^d x_i + r_i) & / & q \rfloor \bmod 2^d \\ (2^d x_i + r_i) & \bmod & q \end{matrix}\right. \quad \text{with } \sum r_i = \tfrac{q+1}{2}$$

Sum all the quantization errors **securely**.

$g_* = \Sigma_i f_i$

Managing the compression error better

Fritzmann *et al.* and Coron *et al.* *reduce* the error to a *negligible* level.

We **keep track** of the compression error and **correct it**.

Compress losslessly by storing the **exact** quantization error for each share x_i.

$$\left.\begin{matrix} y_i \\ f_i \end{matrix}\right\} = \mathrm{DivMod}_d(x) = \left\{ \begin{matrix} \lfloor (2^d x_i + r_i) & / & q \rfloor \bmod 2^d \\ (2^d x_i + r_i) & \bmod & q \end{matrix} \right. \quad \text{with } \sum r_i = \tfrac{q+1}{2}$$

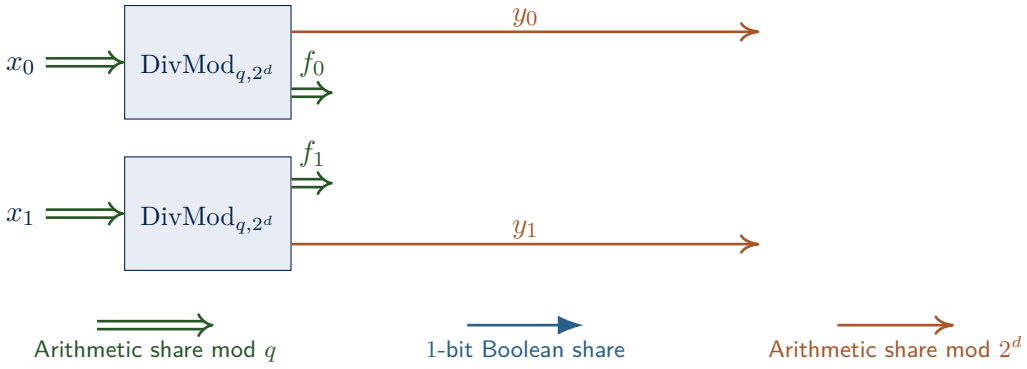
Sum all the quantization errors **securely**.

$g_* = \Sigma_i f_i$

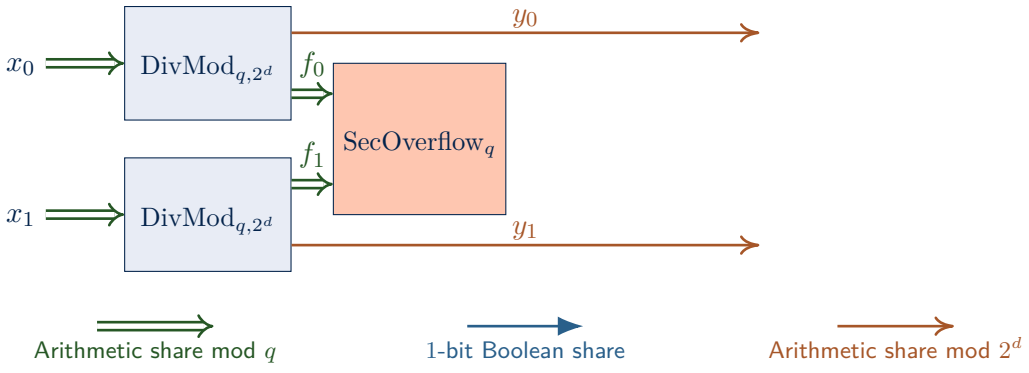
Check if the total error influences the result and correct for it.

$c_* = \lfloor g_*/q \rfloor$; final result is $z_* = y_* + c_* \bmod 2^d$.

Secure 1st-order computation of the correction

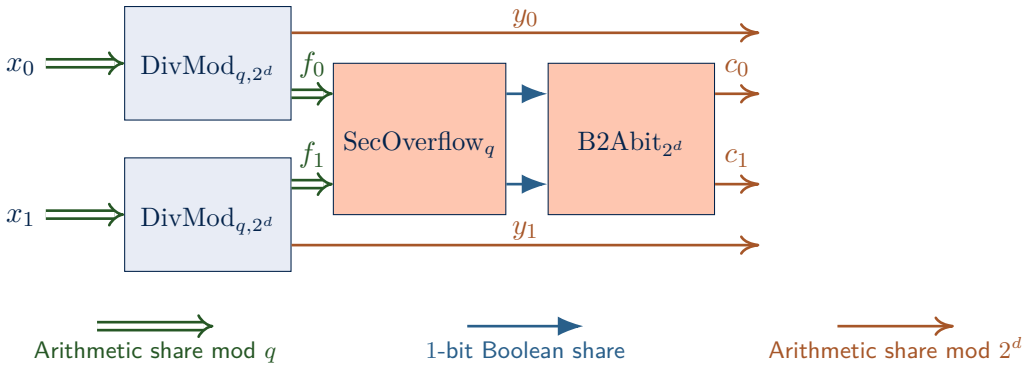


Secure 1st-order computation of the correction



SecOverflow: sum and compare with q using secure addition (SecAdd).

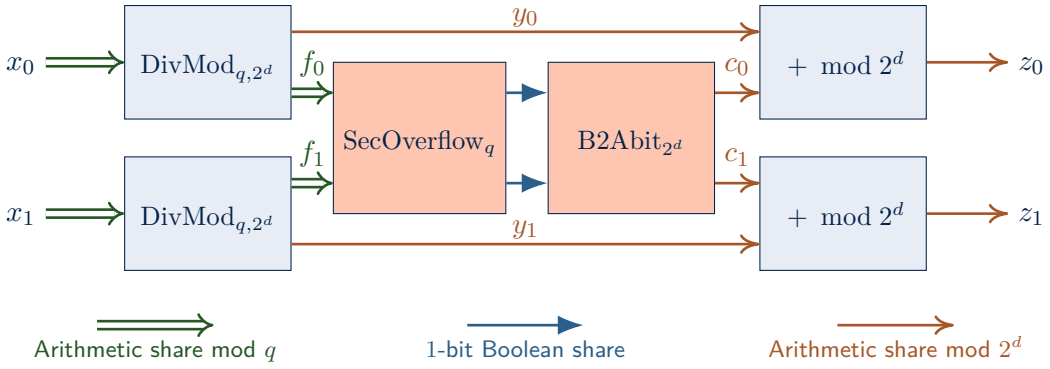
Secure 1st-order computation of the correction



SecOverflow: sum and compare with q using secure addition (SecAdd).

B2Abit: 1-bit Boolean-to-arithmetic conversion

Secure 1st-order computation of the correction



SecOverflow: sum and compare with q using secure addition (SecAdd).

B2Abit: 1-bit Boolean-to-arithmetic conversion

High-order correction function

The correction function for n shares must compute $c = \left\lfloor \frac{\sum f_i}{q} \right\rfloor$

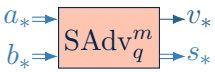
We use a correction tree that sums the f_i pair-wise, comparing the sum with q.
The order of the sharings *grows progressively* with the number of f_i involved.

High-order correction function

The correction function for n shares must compute $c = \left\lfloor \frac{\sum f_i}{q} \right\rfloor$

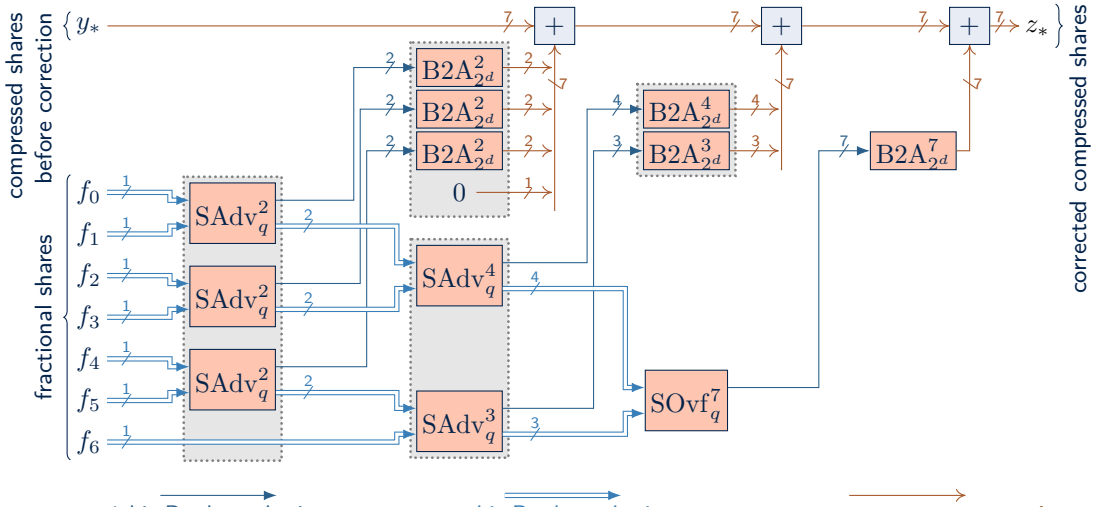
We use a correction tree that sums the f_i pair-wise, comparing the sum with q.
The order of the sharings *grows progressively* with the number of f_i involved.

Secure overflow modulo q, over m shares.
Output: $v = [a + b \geqslant q]$ as a 1-bit Boolean sharing.



Secure addition mod q with modular-overflow output, over m shares.
Outputs: $\begin{cases} v = [a + b \geqslant q] & \text{as a 1-bit Boolean sharing,} \\ s = a + b \bmod q & \text{as a } w\text{-bit Boolean sharing.} \end{cases}$

Secure computation of high-order correction



Comparison with HOCompress [CGMZ23]

Functional comparison

- ≠ Output as arithmetic shares vs. Boolean in previous works
- = Formal proof of security at any order
- ⊕ More generic: arbitrary output modulus, not only 2^d

Comparison with HOCompress [CGMZ23]

Functional comparison

- ≠ Output as arithmetic shares vs. Boolean in previous works
- = Formal proof of security at any order
- ⊕ More generic: arbitrary output modulus, not only 2^d

Performance comparison

Full cost comparison depends on many parameters.

- ⊕ Lower asymptotic complexity, $O(n^2 \log q)$ vs $O(n^2(\log q + \log n))$
- ⊕ Narrower arithmetic operations, better hardware reuse thereof
- ⊖ More costly at low order and/or low d
- ⊕ Less costly at high order and/or high d
- ⊕ Less random bits for $n \geq 3$

Conclusions

Our gadget for masked compression is . . .

- an alternate method to do integer rescaling in a masked way
- based on the exact tracking of quantization error
- competitive with the state of the art
- typically more efficient with high order n and large d

Any question?

Bibliography

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal.
Masking Kyber: First- and higher-order implementations.
IACR TCHES, 2021(4):173–214, 2021.

[CGMZ23] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun.
High-order polynomial comparison and masking lattice-based encryption.
IACR TCHES, 2023(1):153–192, 2023.

[FBR+22] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl.
Masked accelerators and instruction set extensions for post-quantum cryptography.
IACR TCHES, 2022(1):414–460, 2022.

Thank you

life.augmented

Arithmetic operations

The width of the arithmetic operations differs.

Divisions

	Our work	HOCompress
Number of divisions	n	n
Dividend (bits)	$d + \log q$	$d + 2\log q + \log n$
Divisor (bits)	$\log q$	$\log q$
Quotient (bits)	d	$d + \log q + \log n$
Remainder (bits)	$\log q$	unused

Other arithmetic operations are on $\log q (+d)$ bits vs $\log q + d + \log n$ for HOCompress.

Boolean operations

	Our work	HOCompress	HOCompress⋆
Number of operations	Complex dependency on n and d: see Table 1 of the paper for details		
Width (bits)	bitsliced	$d + \log q + \log n$	bitsliced

HOCompress⋆ = HOCompress with bitsliced A2B conversion from Bronchain & Cassiers

Boolean operations

Boolean operations

	Our work	HOCompress	HOCompress⋆
Number of operations	Complex dependency on n and d: see Table 1 of the paper for details		
Width (bits)	bitsliced	$d + \log q + \log n$	bitsliced

HOCompress⋆ = HOCompress with bitsliced A2B conversion from Bronchain & Cassiers

- More bit ops than HOCompress at low order and/or low d
- Less bit ops than HOCompress at high order and/or high d
- Less bit ops than HOCompress⋆ for all cases

Cost comparison – $d = 4$

n	algorithm	random	div. ops	div. size	arith. ops.	Bool. ops
2	SecRescale	164	2	$16 \div 12 \rightarrow 4(12)$	15	$3390 \times 1 = 3390$
	HOCompress	17	2	$29 \div 12 \rightarrow 17$	7	$92 \times 17 = 1564$
	HOCompress*	272	2	$29 \div 12 \rightarrow 17$	7	$4452 \times 1 = 4452$
3	SecRescale	824	3	$16 \div 12 \rightarrow 4(12)$	40	$11992 \times 1 = 11992$
	HOCompress	1044	3	$30 \div 12 \rightarrow 18$	10	$590 \times 18 = 10620$
	HOCompress*	1224	3	$30 \div 12 \rightarrow 18$	10	$16483 \times 1 = 16483$
4	SecRescale	1804	4	$16 \div 12 \rightarrow 4(12)$	69	$23908 \times 1 = 23908$
	HOCompress	2052	4	$30 \div 12 \rightarrow 18$	13	$1066 \times 18 = 19188$
	HOCompress*	2448	4	$30 \div 12 \rightarrow 18$	13	$31024 \times 1 = 31024$
7	SecRescale	7896	7	$16 \div 12 \rightarrow 4(12)$	231	$90684 \times 1 = 90684$
	HOCompress	11153	7	$31 \div 12 \rightarrow 19$	22	$4553 \times 19 = 86507$
	HOCompress*	11286	7	$31 \div 12 \rightarrow 19$	22	$129028 \times 1 = 129028$
8	SecRescale	10840	8	$16 \div 12 \rightarrow 4(12)$	298	$121960 \times 1 = 121960$
	HOCompress	14820	8	$31 \div 12 \rightarrow 19$	25	$5912 \times 19 = 112328$
	HOCompress*	15048	8	$31 \div 12 \rightarrow 19$	25	$169608 \times 1 = 169608$

Cost comparison – $d = 11$

n	algorithm	random	div. ops	div. size	arith. ops.	Bool. ops
2	SecRescale	178	2	$23 \div 12 \to 11(12)$	15	$3390 \times 1 = 3390$
	HOCompress	24	2	$36 \div 12 \to 24$	7	$127 \times 24 = 3048$
	HOCompress*	552	2	$36 \div 12 \to 24$	7	$8974 \times 1 = 8974$
3	SecRescale	880	3	$23 \div 12 \to 11(12)$	40	$11992 \times 1 = 11992$
	HOCompress	1975	3	$37 \div 12 \to 25$	10	$814 \times 25 = 20350$
	HOCompress*	2400	3	$37 \div 12 \to 25$	10	$32170 \times 1 = 32170$
4	SecRescale	1916	4	$23 \div 12 \to 11(12)$	69	$23908 \times 1 = 23908$
	HOCompress	3900	4	$37 \div 12 \to 25$	13	$1472 \times 25 = 36800$
	HOCompress*	4800	4	$37 \div 12 \to 25$	13	$60592 \times 1 = 60592$
7	SecRescale	8358	7	$23 \div 12 \to 11(12)$	231	$90684 \times 1 = 90684$
	HOCompress	20722	7	$38 \div 12 \to 26$	22	$6212 \times 26 = 161512$
	HOCompress*	21450	7	$38 \div 12 \to 26$	22	$244640 \times 1 = 244640$
8	SecRescale	11456	8	$23 \div 12 \to 11(12)$	298	$121960 \times 1 = 121960$
	HOCompress	27560	8	$38 \div 12 \to 26$	25	$8068 \times 26 = 209768$
	HOCompress*	28600	8	$38 \div 12 \to 26$	25	$321648 \times 1 = 321648$