Optimizing Lattice-Based Cryptography

Chitchanok Chuengsatiansup

The University of Adelaide, Australia

17^{th} November 2021

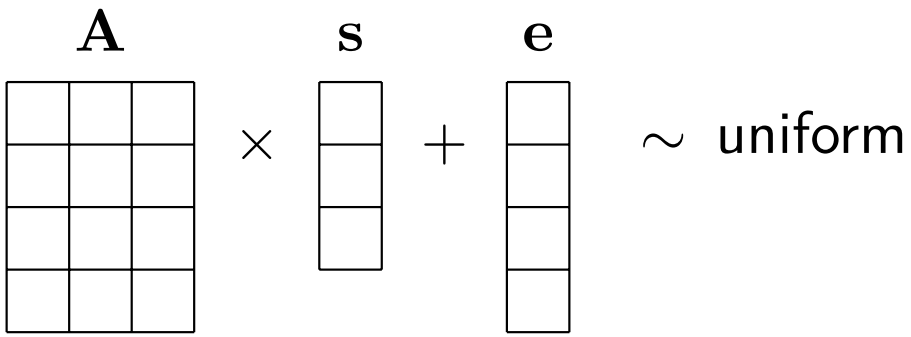
Design and Evaluation for New-generation Cryptography

Hardness assumptions

- Learning-with-Error (LWE)
- NTRU

Hardness assumptions

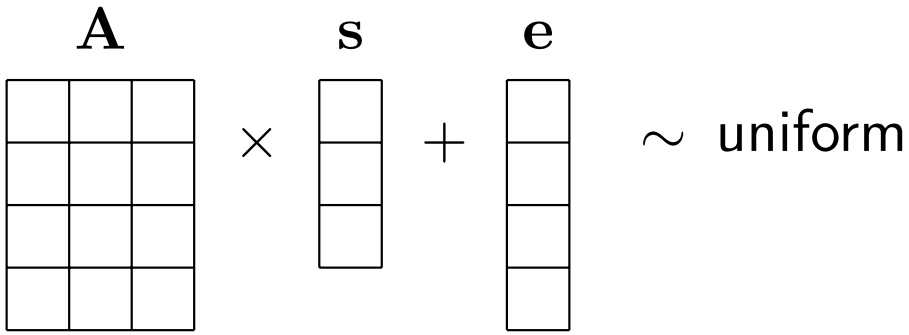
- Learning-with-Error (LWE) $\square \in \mathbb{Z}$



- NTRU

Hardness assumptions

- Learning-with-Error (LWE) $\square \in \mathbb{Z}$



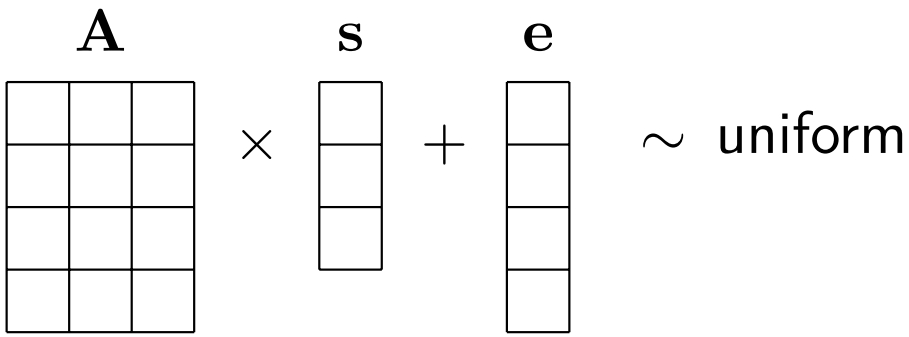
$\sim$ uniform

Learning-with-Rounding (LWR) $\lfloor \mathbf{As} \rfloor \sim$ uniform

- NTRU

Hardness assumptions

- Learning-with-Error (LWE) $\square \in \mathbb{Z}$



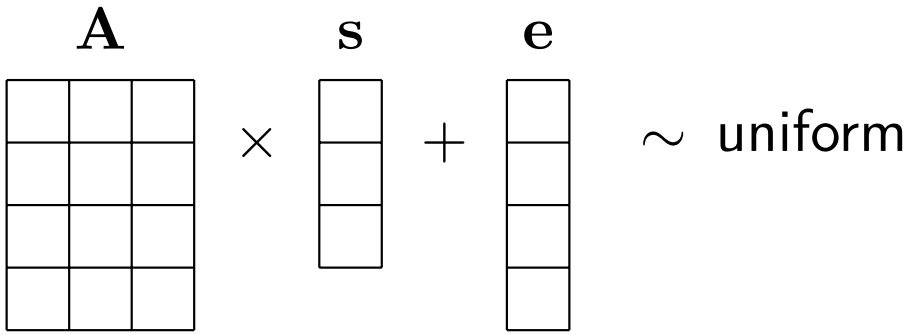
Learning-with-Rounding (LWR) $\lfloor \mathbf{As} \rfloor \sim$ uniform

Ring-LWE/LWR $\square\square\square \in \mathcal{R} = \mathbb{Z}/\Phi$

- NTRU

Hardness assumptions

- Learning-with-Error (LWE) $\square \in \mathbb{Z}$



$\sim$ uniform

Learning-with-Rounding (LWR) $\lfloor \mathbf{As} \rfloor \sim$ uniform

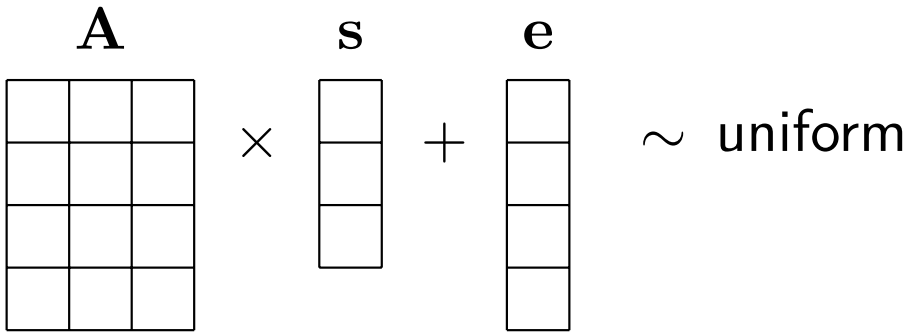
Module-LWE/LWR $\square \in \mathcal{R}$

Ring-LWE/LWR $\square\square\square \in \mathcal{R} = \mathbb{Z}/\Phi$

- NTRU

Hardness assumptions

- Learning-with-Error (LWE) $\square \in \mathbb{Z}$



$\sim$ uniform

Learning-with-Rounding (LWR) $\lfloor \mathbf{As} \rceil \sim$ uniform

Module-LWE/LWR $\square \in \mathcal{R}$

Ring-LWE/LWR $\square\square\square \in \mathcal{R} = \mathbb{Z}/\Phi$

- NTRU

$f, g \in \mathcal{R}$: $f/g \sim$ uniform

Multiplication algorithms

- Schoolbook
- Karatsuba
- Toom–Cook
- Number-theoretic transform (NTT)
- etc.

Multiplication algorithms

- Schoolbook
- Karatsuba
- Toom–Cook
- Number-theoretic transform (NTT)
  - restrictions on modulus /polynomial
- etc.

Multiplication algorithms

- Schoolbook
- Karatsuba
- Toom–Cook

} generic

- Number-theoretic transform (NTT)
  - restrictions on modulus /polynomial
- etc.

Schoolbook multiplication

- Compute $H = FG$

Schoolbook multiplication

- Compute $H = FG$

  example:
  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

Schoolbook multiplication

- Compute $H = FG$

  example:
  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

$$
\begin{aligned}
h_0 &= f_0g_0 \\
h_1 &= f_0g_1 + f_1g_0 \\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 \\
h_4 &= f_1g_3 + f_2g_2 + f_3g_1 \\
h_5 &= f_2g_3 + f_3g_2 \\
h_6 &= f_3g_3
\end{aligned}
$$

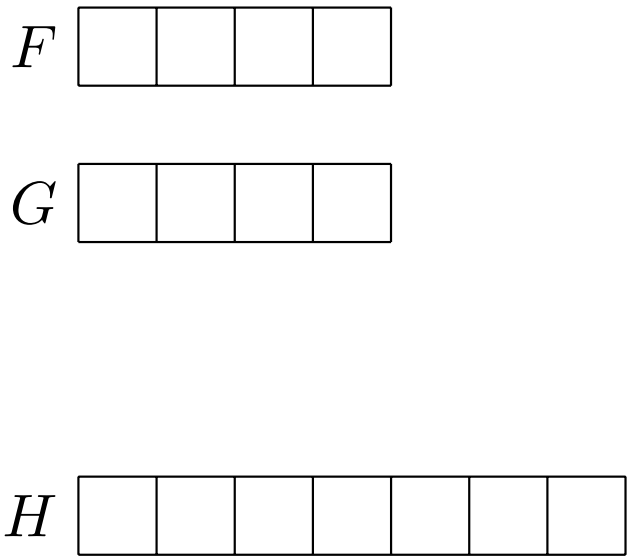
Schoolbook multiplication

- Compute $H = FG$

  example:

  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

$$
\begin{aligned}
h_0 &= f_0g_0 \\
h_1 &= f_0g_1 + f_1g_0 \\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 \\
h_4 &= f_1g_3 + f_2g_2 + f_3g_1 \\
h_5 &= f_2g_3 + f_3g_2 \\
h_6 &= f_3g_3
\end{aligned}
$$



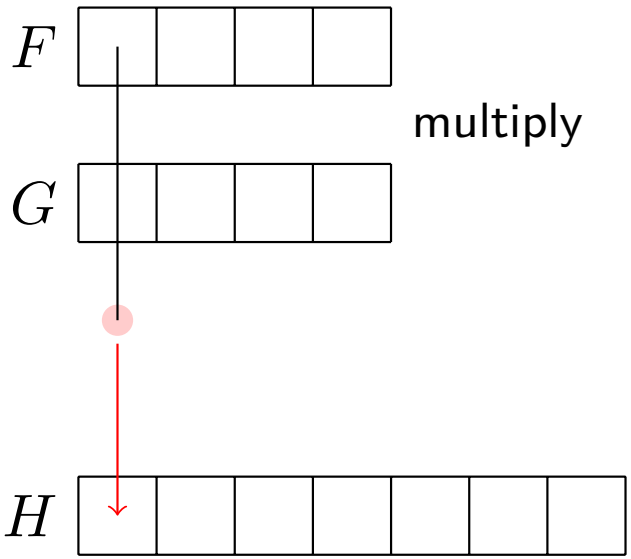
Schoolbook multiplication

- Compute $H = FG$

  example:

  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

$$
\begin{aligned}
h_0 &= f_0g_0 \\
h_1 &= f_0g_1 + f_1g_0 \\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 \\
h_4 &= f_1g_3 + f_2g_2 + f_3g_1 \\
h_5 &= f_2g_3 + f_3g_2 \\
h_6 &= f_3g_3
\end{aligned}
$$



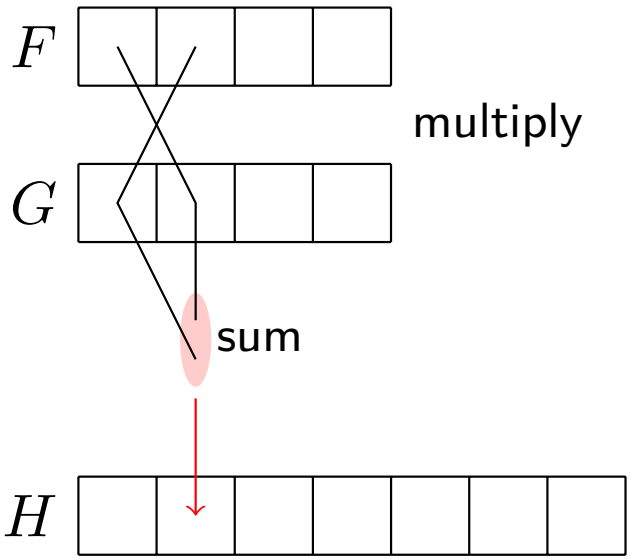
Schoolbook multiplication

- Compute $H = FG$

  example:

  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

$$
\begin{aligned}
h_0 &= f_0g_0 \\
h_1 &= f_0g_1 + f_1g_0 \\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 \\
h_4 &= f_1g_3 + f_2g_2 + f_3g_1 \\
h_5 &= f_2g_3 + f_3g_2 \\
h_6 &= f_3g_3
\end{aligned}
$$



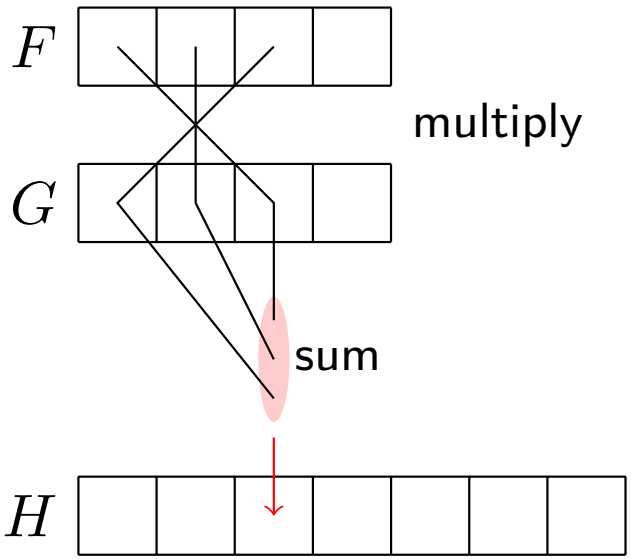
Schoolbook multiplication

- Compute $H = FG$

  example:

  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

$$
\begin{aligned}
h_0 &= f_0g_0 \\
h_1 &= f_0g_1 + f_1g_0 \\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 \\
h_4 &= f_1g_3 + f_2g_2 + f_3g_1 \\
h_5 &= f_2g_3 + f_3g_2 \\
h_6 &= f_3g_3
\end{aligned}
$$



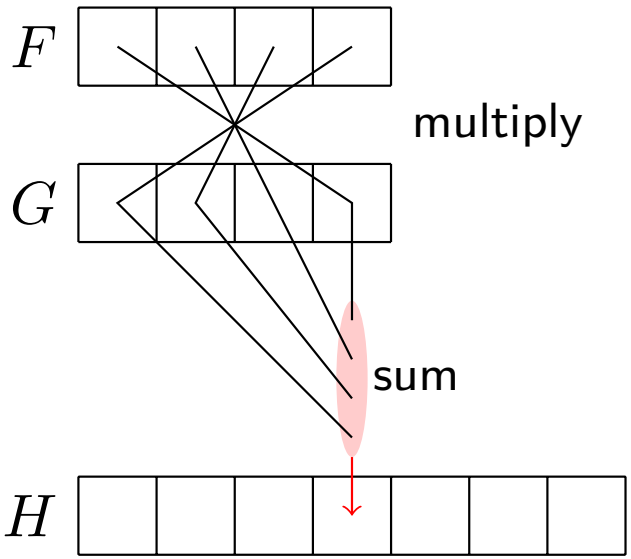
Schoolbook multiplication

- Compute $H = FG$

  example:

  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

$$
\begin{aligned}
h_0 &= f_0g_0 \\
h_1 &= f_0g_1 + f_1g_0 \\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 \\
h_4 &= f_1g_3 + f_2g_2 + f_3g_1 \\
h_5 &= f_2g_3 + f_3g_2 \\
h_6 &= f_3g_3
\end{aligned}
$$



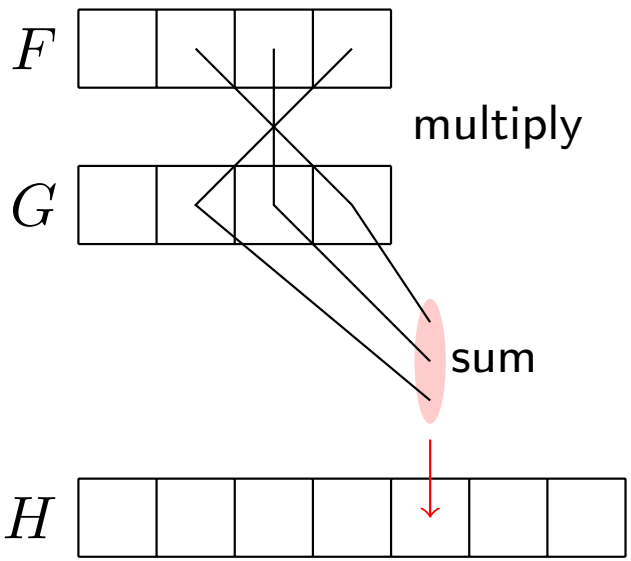
Schoolbook multiplication

- Compute $H = FG$

  example:

  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

$$
\begin{aligned}
h_0 &= f_0g_0 \\
h_1 &= f_0g_1 + f_1g_0 \\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 \\
h_4 &= f_1g_3 + f_2g_2 + f_3g_1 \\
h_5 &= f_2g_3 + f_3g_2 \\
h_6 &= f_3g_3
\end{aligned}
$$



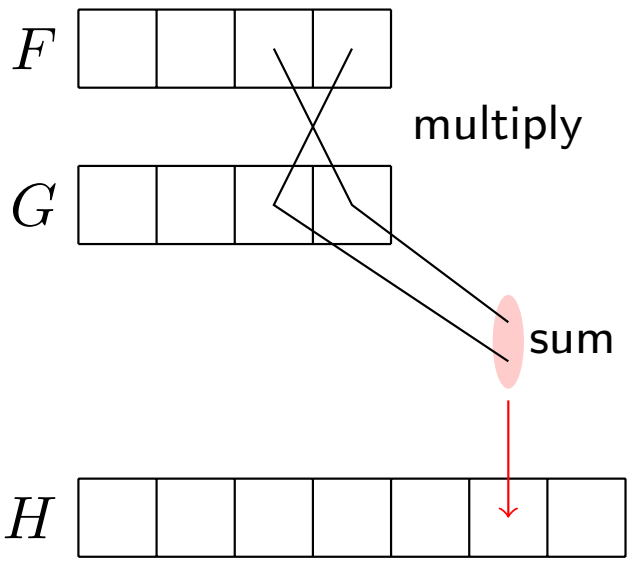
Schoolbook multiplication

- Compute $H = FG$

  example:

  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

$$
\begin{aligned}
h_0 &= f_0g_0 \\
h_1 &= f_0g_1 + f_1g_0 \\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 \\
h_4 &= f_1g_3 + f_2g_2 + f_3g_1 \\
h_5 &= f_2g_3 + f_3g_2 \\
h_6 &= f_3g_3
\end{aligned}
$$



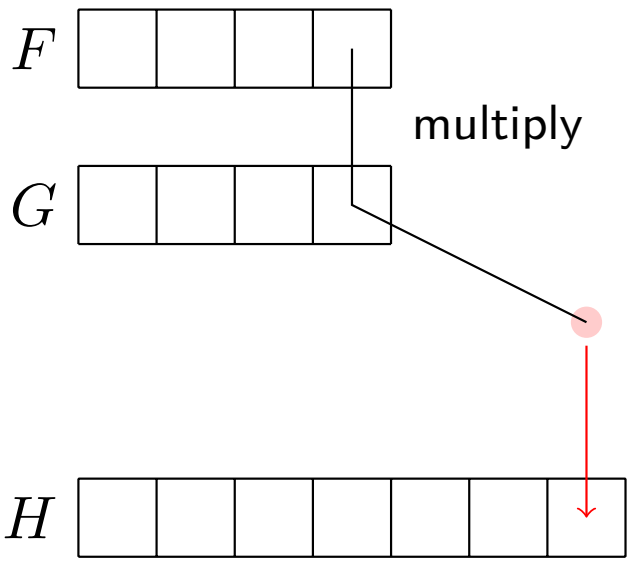
Schoolbook multiplication

- Compute $H = FG$

  example:

  $F = f_0 + f_1x^1 + f_2x^2 + f_3x^3; \quad G = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

$$
\begin{aligned}
h_0 &= f_0g_0 \\
h_1 &= f_0g_1 + f_1g_0 \\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 \\
h_4 &= f_1g_3 + f_2g_2 + f_3g_1 \\
h_5 &= f_2g_3 + f_3g_2 \\
h_6 &= f_3g_3
\end{aligned}
$$



Karatsuba multiplication

$$(F_0 + F_1 x^n)(G_0 + G_1 x^n) = (1 - x^n)(F_0 G_0 - F_1 G_1 x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

Karatsuba multiplication

$$(F_0 + F_1 x^n)(G_0 + G_1 x^n) = (1 - x^n)(F_0 G_0 - F_1 G_1 x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$\begin{aligned} F_0 &= f_0 + f_1 x + f_2 x^2 + \ldots + f_{63} x^{63}; & F_1 &= f_{64} + f_{65} x + f_{66} x^2 + \ldots + f_{127} x^{63}; \\ G_0 &= g_0 + g_1 x + g_2 x^2 + \ldots + g_{63} x^{63}; & G_1 &= g_{64} + g_{65} x + g_{66} x^2 + \ldots + g_{127} x^{63}; \end{aligned}$$

$$FG = (1 - x^{64})(F_0 G_0 - F_1 G_1 x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$

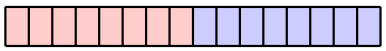
Karatsuba multiplication

$$(F_0 + F_1x^n)(G_0 + G_1x^n) = (1 - x^n)(F_0G_0 - F_1G_1x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$\begin{aligned} F_0 &= f_0 + f_1x + f_2x^2 + \ldots + f_{63}x^{63}; & F_1 &= f_{64} + f_{65}x + f_{66}x^2 + \ldots + f_{127}x^{63}; \\ G_0 &= g_0 + g_1x + g_2x^2 + \ldots + g_{63}x^{63}; & G_1 &= g_{64} + g_{65}x + g_{66}x^2 + \ldots + g_{127}x^{63}; \end{aligned}$$

$$FG = (1 - x^{64})(F_0G_0 - F_1G_1x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$

F  $\times$  G

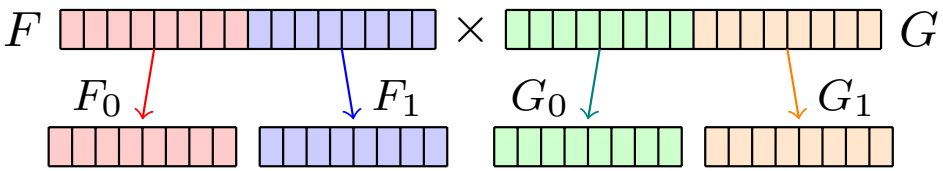
Karatsuba multiplication

$$(F_0 + F_1 x^n)(G_0 + G_1 x^n) = (1 - x^n)(F_0 G_0 - F_1 G_1 x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$F_0 = f_0 + f_1 x + f_2 x^2 + \ldots + f_{63} x^{63}; \quad F_1 = f_{64} + f_{65} x + f_{66} x^2 + \ldots + f_{127} x^{63};$$
$$G_0 = g_0 + g_1 x + g_2 x^2 + \ldots + g_{63} x^{63}; \quad G_1 = g_{64} + g_{65} x + g_{66} x^2 + \ldots + g_{127} x^{63};$$

$$FG = (1 - x^{64})(F_0 G_0 - F_1 G_1 x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



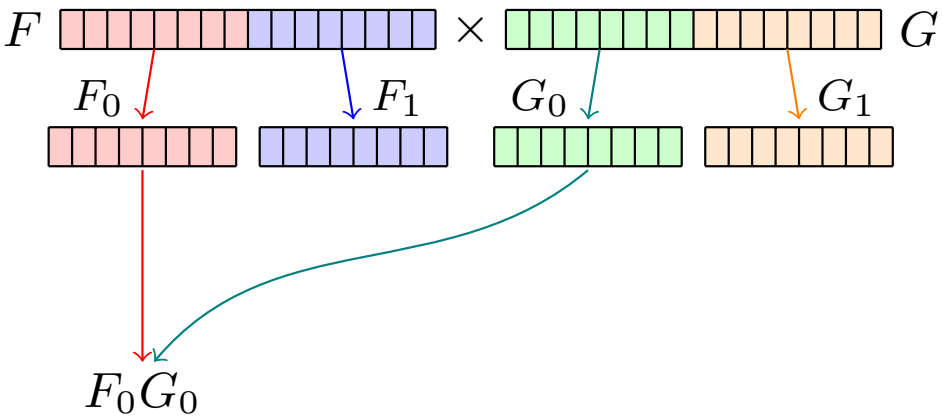
Karatsuba multiplication

$$(F_0 + F_1 x^n)(G_0 + G_1 x^n) = (1 - x^n)(F_0 G_0 - F_1 G_1 x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$F_0 = f_0 + f_1 x + f_2 x^2 + \ldots + f_{63} x^{63}; \quad F_1 = f_{64} + f_{65} x + f_{66} x^2 + \ldots + f_{127} x^{63};$$
$$G_0 = g_0 + g_1 x + g_2 x^2 + \ldots + g_{63} x^{63}; \quad G_1 = g_{64} + g_{65} x + g_{66} x^2 + \ldots + g_{127} x^{63};$$

$$FG = (1 - x^{64})(F_0 G_0 - F_1 G_1 x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



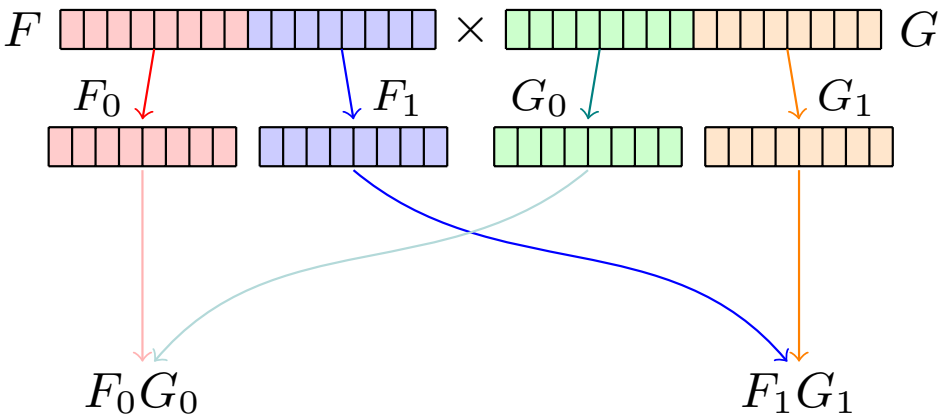
Karatsuba multiplication

$$(F_0 + F_1 x^n)(G_0 + G_1 x^n) = (1 - x^n)(F_0 G_0 - F_1 G_1 x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$F_0 = f_0 + f_1 x + f_2 x^2 + \ldots + f_{63} x^{63}; \quad F_1 = f_{64} + f_{65} x + f_{66} x^2 + \ldots + f_{127} x^{63};$$
$$G_0 = g_0 + g_1 x + g_2 x^2 + \ldots + g_{63} x^{63}; \quad G_1 = g_{64} + g_{65} x + g_{66} x^2 + \ldots + g_{127} x^{63};$$

$$FG = (1 - x^{64})(F_0 G_0 - F_1 G_1 x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



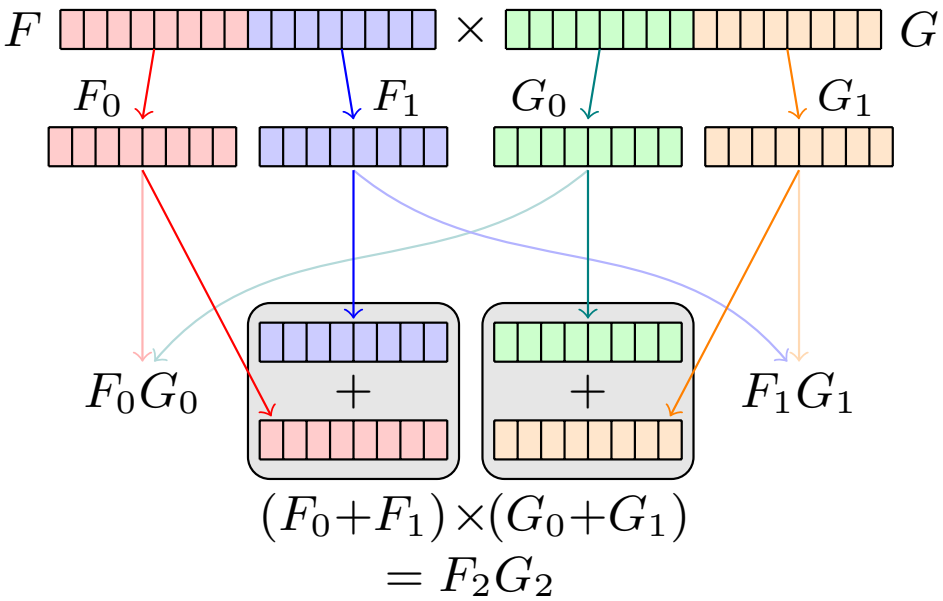
Karatsuba multiplication

$$(F_0 + F_1 x^n)(G_0 + G_1 x^n) = (1 - x^n)(F_0 G_0 - F_1 G_1 x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$F_0 = f_0 + f_1 x + f_2 x^2 + \ldots + f_{63} x^{63}; \quad F_1 = f_{64} + f_{65} x + f_{66} x^2 + \ldots + f_{127} x^{63};$$
$$G_0 = g_0 + g_1 x + g_2 x^2 + \ldots + g_{63} x^{63}; \quad G_1 = g_{64} + g_{65} x + g_{66} x^2 + \ldots + g_{127} x^{63};$$

$$FG = (1 - x^{64})(F_0 G_0 - F_1 G_1 x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



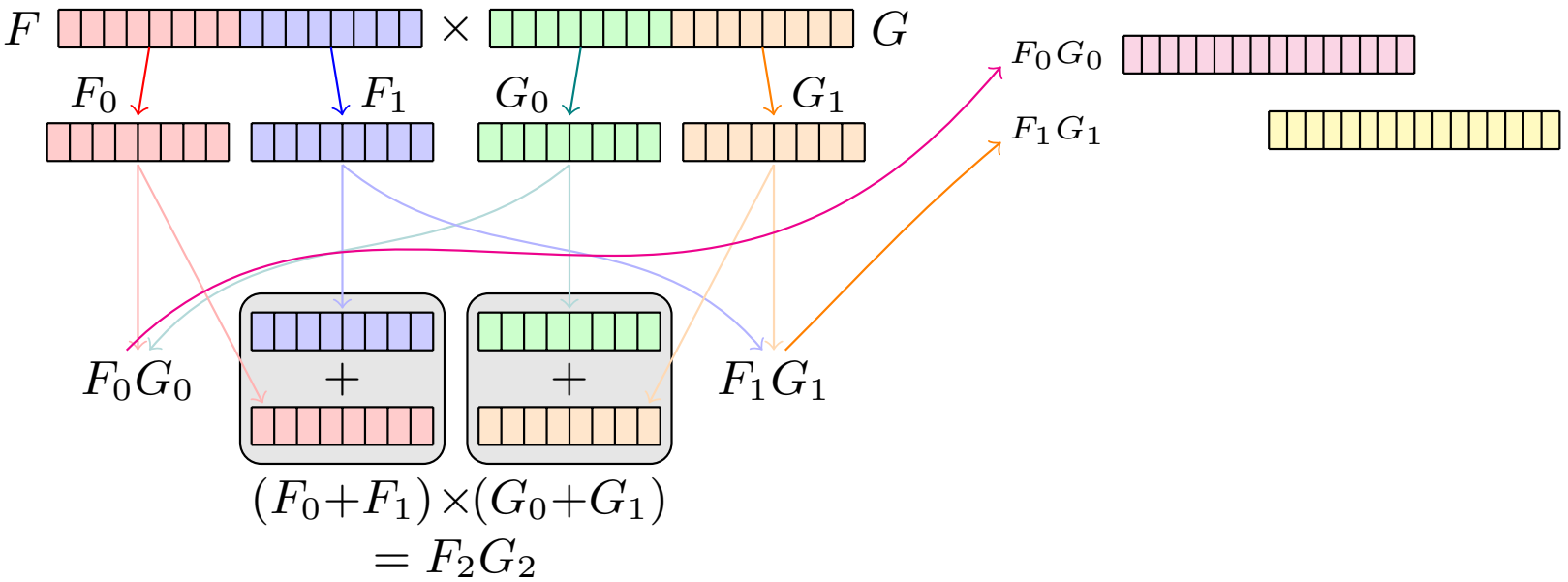
Karatsuba multiplication

$$(F_0 + F_1 x^n)(G_0 + G_1 x^n) = (1 - x^n)(F_0 G_0 - F_1 G_1 x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$F_0 = f_0 + f_1 x + f_2 x^2 + \ldots + f_{63} x^{63}; \quad F_1 = f_{64} + f_{65} x + f_{66} x^2 + \ldots + f_{127} x^{63};$$
$$G_0 = g_0 + g_1 x + g_2 x^2 + \ldots + g_{63} x^{63}; \quad G_1 = g_{64} + g_{65} x + g_{66} x^2 + \ldots + g_{127} x^{63};$$

$$FG = (1 - x^{64})(F_0 G_0 - F_1 G_1 x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



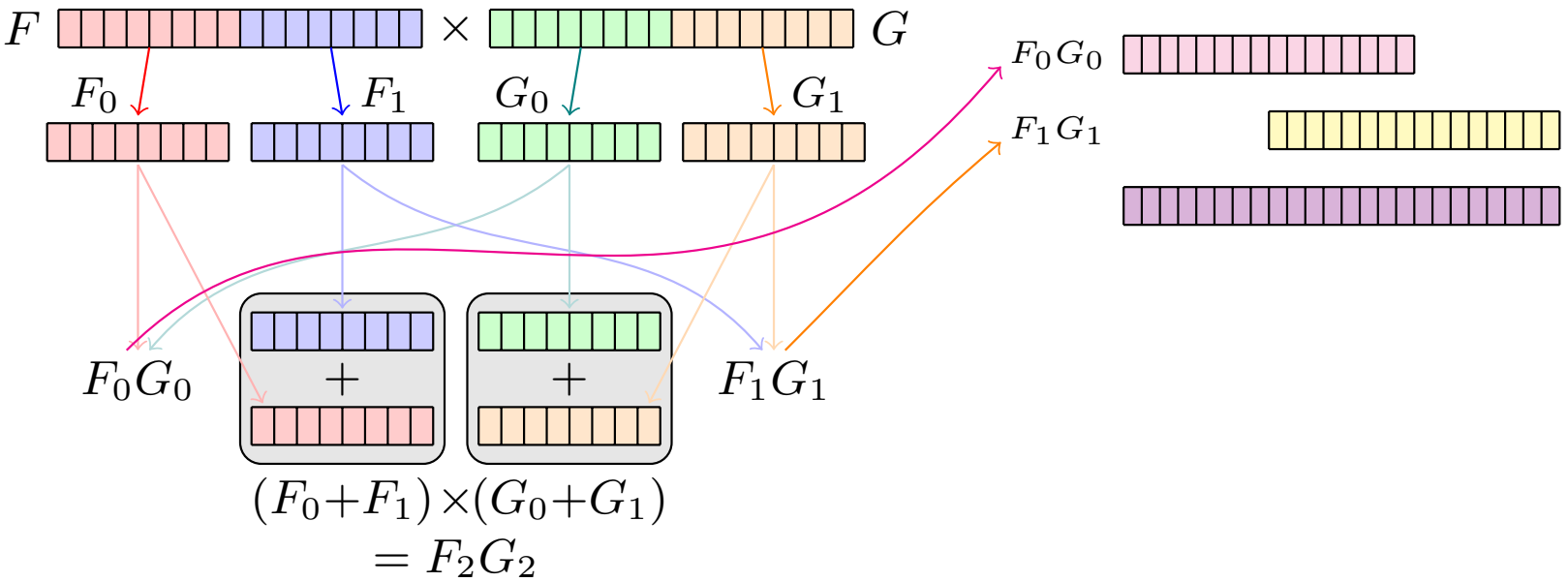
Karatsuba multiplication

$$(F_0 + F_1x^n)(G_0 + G_1x^n) = (1 - x^n)(F_0G_0 - F_1G_1x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$F_0 = f_0 + f_1x + f_2x^2 + \ldots + f_{63}x^{63}; \quad F_1 = f_{64} + f_{65}x + f_{66}x^2 + \ldots + f_{127}x^{63};$$
$$G_0 = g_0 + g_1x + g_2x^2 + \ldots + g_{63}x^{63}; \quad G_1 = g_{64} + g_{65}x + g_{66}x^2 + \ldots + g_{127}x^{63};$$

$$FG = (1 - x^{64})(F_0G_0 - F_1G_1x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



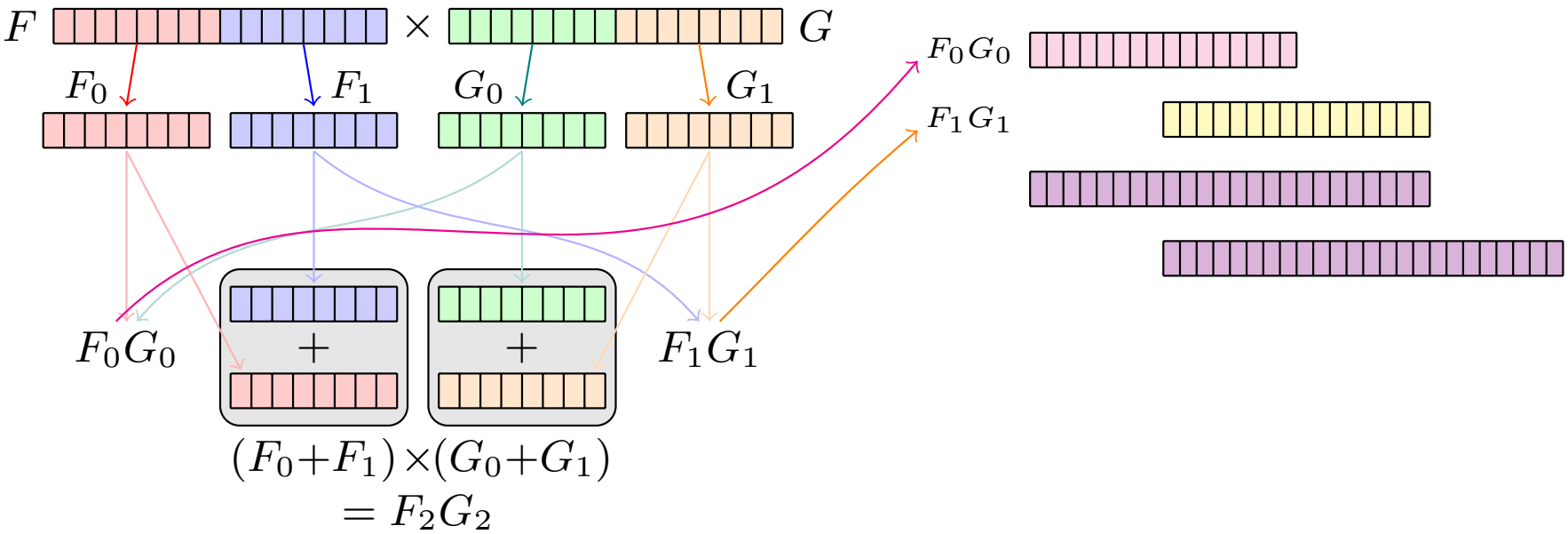
Karatsuba multiplication

$$(F_0 + F_1x^n)(G_0 + G_1x^n) = (1 - x^n)(F_0G_0 - F_1G_1x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$F_0 = f_0 + f_1x + f_2x^2 + \ldots + f_{63}x^{63}; \quad F_1 = f_{64} + f_{65}x + f_{66}x^2 + \ldots + f_{127}x^{63};$$
$$G_0 = g_0 + g_1x + g_2x^2 + \ldots + g_{63}x^{63}; \quad G_1 = g_{64} + g_{65}x + g_{66}x^2 + \ldots + g_{127}x^{63};$$

$$FG = (1 - x^{64})(F_0G_0 - F_1G_1x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



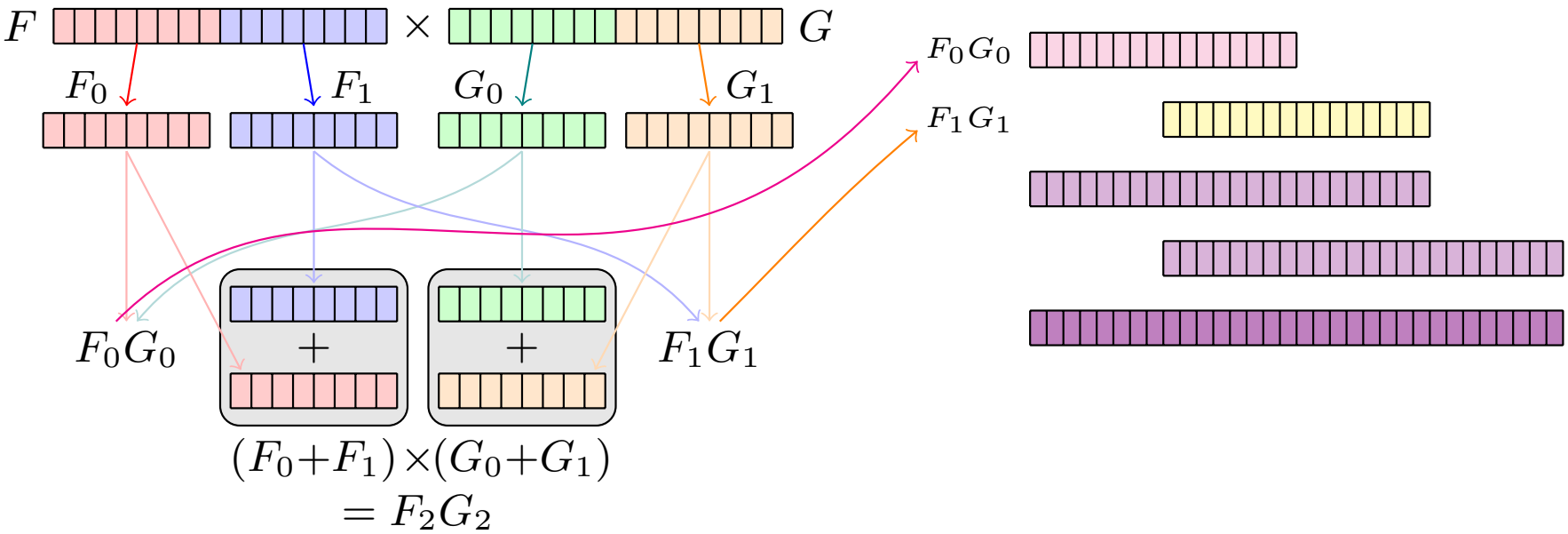
Karatsuba multiplication

$$(F_0 + F_1x^n)(G_0 + G_1x^n) = (1 - x^n)(F_0G_0 - F_1G_1x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$F_0 = f_0 + f_1x + f_2x^2 + \ldots + f_{63}x^{63}; \quad F_1 = f_{64} + f_{65}x + f_{66}x^2 + \ldots + f_{127}x^{63};$$
$$G_0 = g_0 + g_1x + g_2x^2 + \ldots + g_{63}x^{63}; \quad G_1 = g_{64} + g_{65}x + g_{66}x^2 + \ldots + g_{127}x^{63};$$

$$FG = (1 - x^{64})(F_0G_0 - F_1G_1x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



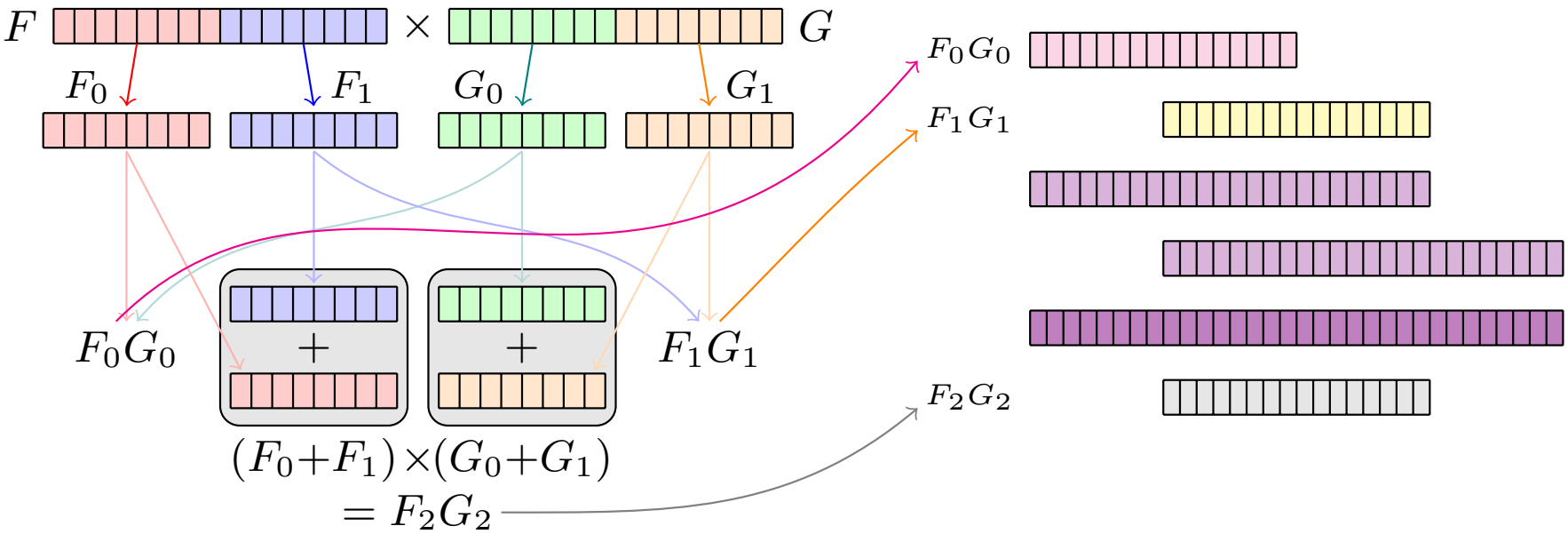
Karatsuba multiplication

$$(F_0 + F_1 x^n)(G_0 + G_1 x^n) = (1 - x^n)(F_0 G_0 - F_1 G_1 x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

$$F_0 = f_0 + f_1 x + f_2 x^2 + \ldots + f_{63} x^{63}; \quad F_1 = f_{64} + f_{65} x + f_{66} x^2 + \ldots + f_{127} x^{63};$$
$$G_0 = g_0 + g_1 x + g_2 x^2 + \ldots + g_{63} x^{63}; \quad G_1 = g_{64} + g_{65} x + g_{66} x^2 + \ldots + g_{127} x^{63};$$

$$FG = (1 - x^{64})(F_0 G_0 - F_1 G_1 x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



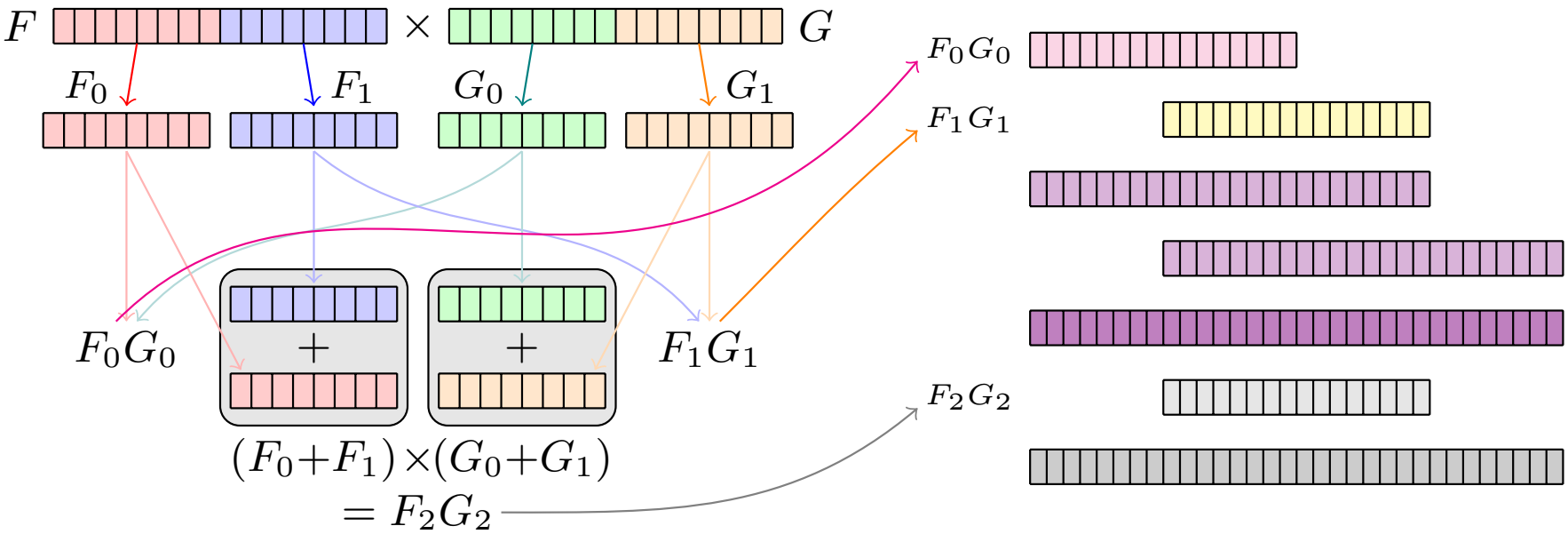
Karatsuba multiplication

$$(F_0 + F_1x^n)(G_0 + G_1x^n) = (1 - x^n)(F_0G_0 - F_1G_1x^n) + (F_0 + F_1)(G_0 + G_1)x^n$$

- example: $2n = 128$

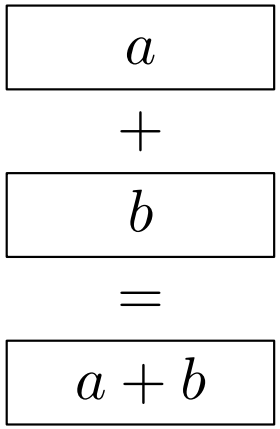
$$F_0 = f_0 + f_1x + f_2x^2 + \ldots + f_{63}x^{63}; \quad F_1 = f_{64} + f_{65}x + f_{66}x^2 + \ldots + f_{127}x^{63};$$
$$G_0 = g_0 + g_1x + g_2x^2 + \ldots + g_{63}x^{63}; \quad G_1 = g_{64} + g_{65}x + g_{66}x^2 + \ldots + g_{127}x^{63};$$

$$FG = (1 - x^{64})(F_0G_0 - F_1G_1x^{64}) + (F_0 + F_1)(G_0 + G_1)x^{64}$$



Vectorization speedups

without vector



Vectorization speedups

without vector

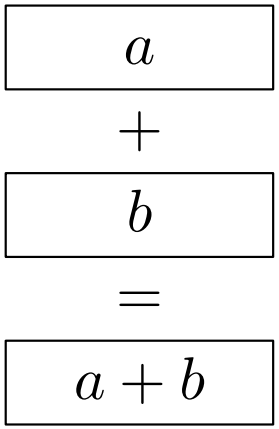
a
$+$
b
$=$
$a + b$

with vector

a_0	a_1	a_2	a_3
$+$	$+$	$+$	$+$
b_0	b_1	b_2	b_3
$=$	$=$	$=$	$=$
$a_0 + b_0$	$a_1 + b_1$	$a_2 + b_2$	$a_3 + b_3$

Vectorization speedups

without vector



with vector

a_0	a_1	a_2	a_3
+	+	+	+
b_0	b_1	b_2	b_3
=	=	=	=
$a_0 + b_0$	$a_1 + b_1$	$a_2 + b_2$	$a_3 + b_3$

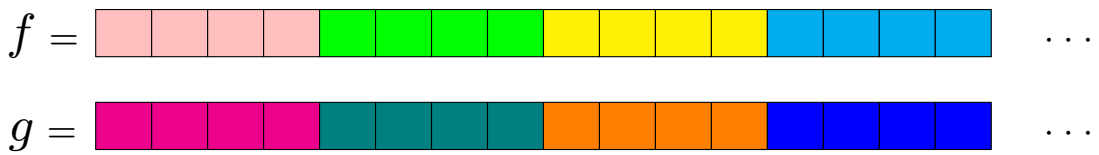
- **single** instruction performing n **independent** operations on **aligned** inputs

Vectorizing multiplication

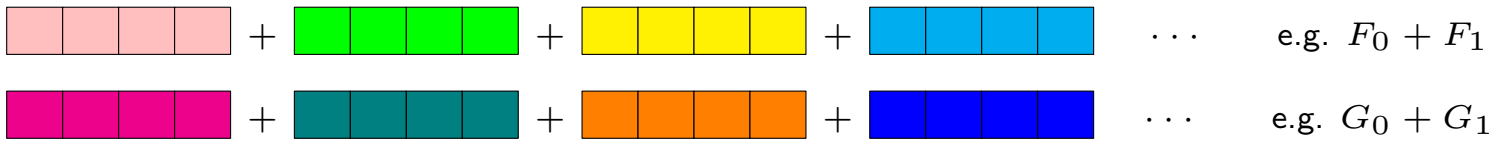
$f =$  $\cdots$

$g =$  $\cdots$

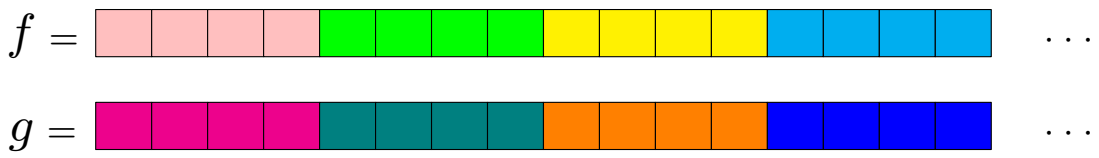
Vectorizing multiplication



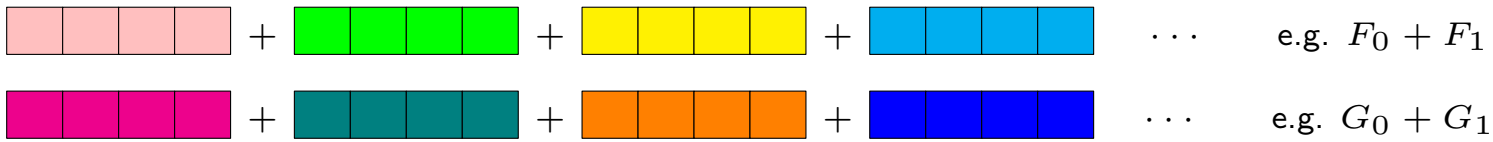
- Karatsuba
  - vectorize inside each limb



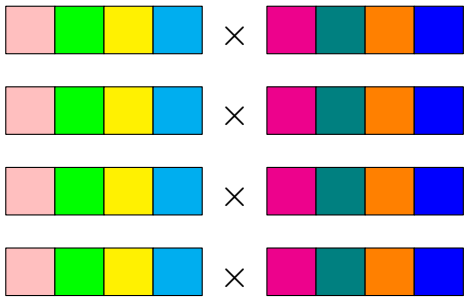
Vectorizing multiplication



- Karatsuba
  - vectorize inside each limb



- Schoolbook
  - transpose inputs
  - vectorize *across* independent multiplications



$$
\begin{aligned}
h_0 &= f_0 g_0 \\
h_1 &= f_0 g_1 + f_1 g_0 \\
h_2 &= f_0 g_2 + f_1 g_1 + f_2 g_0 \\
h_3 &= f_0 g_3 + f_1 g_2 + f_2 g_1 + f_3 g_0 \\
h_4 &= f_1 g_3 + f_2 g_2 + f_3 g_1 \\
h_5 &= f_2 g_3 + f_3 g_2 \\
h_6 &= f_3 g_3
\end{aligned}
$$

Optimization: implementation perspective

- Parameter selection: $p, q, m, n, d, \chi, \ldots$
- Error vs rounding
- LWE/LWR: plain vs module vs ring
- Multiplication algorithms
- Speed vs memory vs security
- Side-channel security
- etc.

PRF from (M)LWR

Based on

Towards practical GGM-based PRF from
(Module-) Learning-with-Rounding
Joint work with Damien Stehlé

Practical and provably secure lattice-based pseudorandom functions
constructed using GGM design
achieving at least 128-bit post-quantum security

Pseudorandomness

Focus on deterministic polynomial-time algorithms whose outputs look uniformly random

Pseudorandomness

Focus on deterministic polynomial-time algorithms whose outputs look uniformly random

- Pseudorandom generator (PRG)

$$G : \{0,1\}^{n_1} \rightarrow \{0,1\}^{n_2} \quad \text{with } n_1 \leq n_2$$

Pseudorandomness

Focus on deterministic polynomial-time algorithms whose outputs look uniformly random

- Pseudorandom generator (PRG)

$$G : \{0,1\}^{n_1} \rightarrow \{0,1\}^{n_2} \quad \text{with } n_1 \leq n_2$$

- Pseudorandom function (PRF)

$$F : \{0,1\}^{k} \times \{0,1\}^{x} \rightarrow \{0,1\}^{n}$$
$$F_k : \{0,1\}^{x} \rightarrow \{0,1\}^{n}$$

Lattice-based construction

- Post-quantum candidate
- Worst-case-to-average-case reductions
- Fast arithmetic

Comparison

schemes	assumptions	mod	key	pp.	mul
SPRING-BCH [1]	R $\lfloor$sub prod$\rceil$	257	16396	129	115
SPRING-CRT [1]	R $\lfloor$sub prod$\rceil$	514	18444	145	115
[2] + AKPW13, R[3]	RLWR, GGM	$> 2^{40}$	655360	5120	5243*
BPR12,syn,R [2]	RLWR, syn	$> 2^{128}$	163840	-	10486*
BPR12,direct,R [2]	R $\lfloor$sub prod$\rceil$	$> 2^{128}$	2097152	16384	10486*
[2] + AKPW13 [3]	LWR, GGM	$> 2^{40}$	313600000	3500	250880*
BPR12,direct [2]	$\lfloor$sub prod$\rceil$	$> 2^{128}$	1003520000	11200	501760*
BPR12,GGM [2]	LWR, GGM	$> 2^{128}$	11200	31360000	1003520*
BPR12,GGM,R [2]	RLWR, GGM	$> 2^{128}$	16384	67108864	42949673*
BPR12,syn [2]	LWR, syn	$> 2^{128}$	74097496	-	351232000*

- key and public parameter (pp.) are of size in bytes
- multiplication (mul) is considered only matrix multiplication and measured in terms of thousand 16-bit-by-16-bit integer multiplications
- $*$ means extrapolated cost
- comparison includes lattice-based PRFs that provide at least 128-bit security

[1] A.Banerjee,H.Brenner,G.Leurent,C.Peikert,A.Rosen, *"SPRING: Fast Pseudorandom Functions from Rounded Ring Products"*.
[2] A.Banerjee,C.Peikert,A.Rosen, *"Pseudorandom Functions and Lattices"*.
[3] J.Alwen,S.Krenn,K.Pietrzak,D.Wichs, *"Learning with Rounding, Revisited - New Reduction, Properties and Applications"*.

Comparison

schemes	assumptions	mod	key	pp.	mul
SPRING-BCH [1]	R $\lfloor$sub prod$\rceil$	257	16396	129	115
SPRING-CRT [1]	R $\lfloor$sub prod$\rceil$	514	18444	145	115
Our scheme, M	MLWR, GGM	2^{16}	192	24576	1991
Our scheme	LWR, GGM	2^{16}	200	2944000	2944
[2] + AKPW13, R[3]	RLWR, GGM	$> 2^{40}$	655360	5120	5243*
BPR12,syn,R [2]	RLWR, syn	$> 2^{128}$	163840	-	10486*
BPR12,direct,R [2]	R $\lfloor$sub prod$\rceil$	$> 2^{128}$	2097152	16384	10486*
[2] + AKPW13 [3]	LWR, GGM	$> 2^{40}$	313600000	3500	250880*
BPR12,direct [2]	$\lfloor$sub prod$\rceil$	$> 2^{128}$	1003520000	11200	501760*
BPR12,GGM [2]	LWR, GGM	$> 2^{128}$	11200	31360000	1003520*
BPR12,GGM,R [2]	RLWR, GGM	$> 2^{128}$	16384	67108864	42949673*
BPR12,syn [2]	LWR, syn	$> 2^{128}$	74097496	-	351232000*

- key and public parameter (pp.) are of size in bytes
- multiplication (mul) is considered only matrix multiplication and measured in terms of thousand 16-bit-by-16-bit integer multiplications
- $*$ means extrapolated cost
- comparison includes lattice-based PRFs that provide at least 128-bit security

[1] A.Banerjee,H.Brenner,G.Leurent,C.Peikert,A.Rosen, *"SPRING: Fast Pseudorandom Functions from Rounded Ring Products"*.
[2] A.Banerjee,C.Peikert,A.Rosen, *"Pseudorandom Functions and Lattices"*.
[3] J.Alwen,S.Krenn,K.Pietrzak,D.Wichs, *"Learning with Rounding, Revisited - New Reduction, Properties and Applications"*.

GGM-based PRF

GGM construction [GGM84]

GGM-based PRF

GGM construction [GGM84]

- requires length-doubling PRG

$$G : \{0,1\}^{n_1} \rightarrow \{0,1\}^{n_2} \quad \text{where } 2 \cdot n_1 = n_2$$

GGM-based PRF

GGM construction [GGM84]

- requires length-doubling PRG

$$G : \{0,1\}^{n_1} \rightarrow \{0,1\}^{n_2} \quad \text{where } 2 \cdot n_1 = n_2$$

- defines as

$$F_k(x_1 x_2 \cdots x_\ell) = G_{x_\ell}(\cdots(G_{x_2}(G_{x_1}(k)))\cdots)$$

where $G(\cdot) = G_0(\cdot)||G_1(\cdot)$

GGM-based PRF

GGM construction [GGM84]

- requires length-doubling PRG

$$G : \{0,1\}^{n_1} \rightarrow \{0,1\}^{n_2} \quad \text{where } 2 \cdot n_1 = n_2$$

- defines as

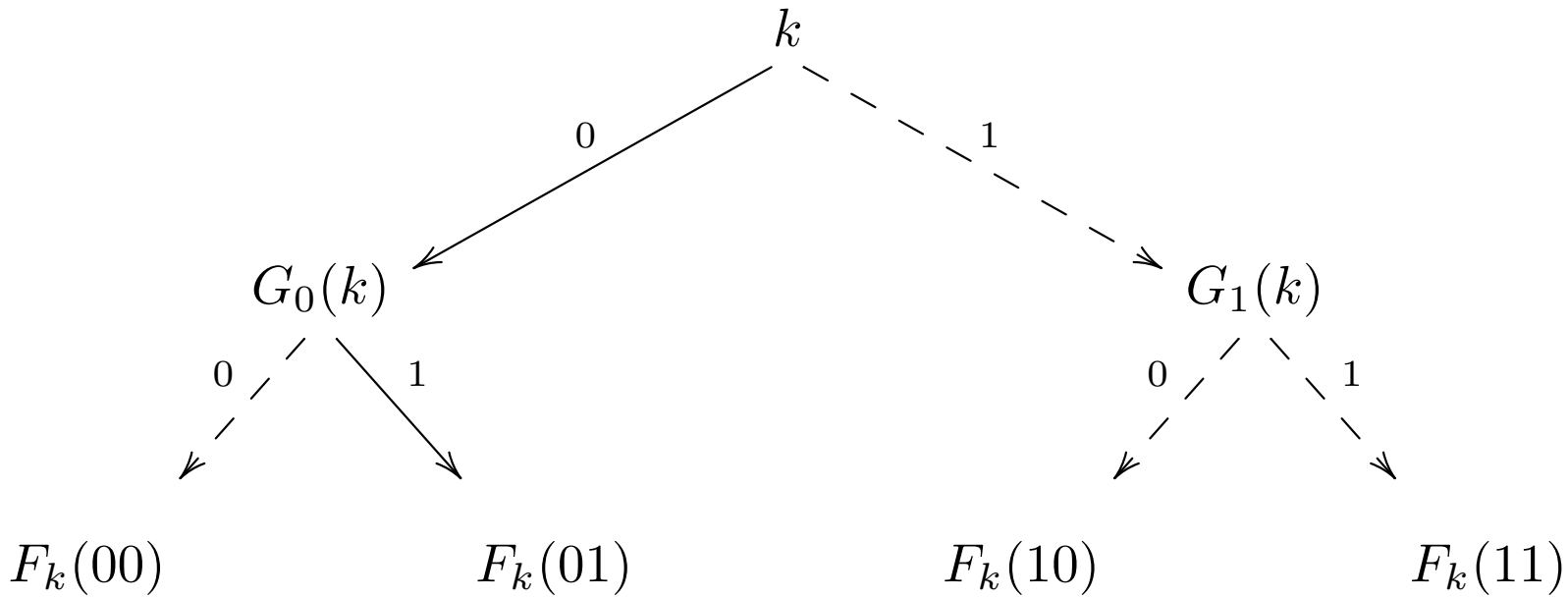
$$F_k(x_1 x_2 \cdots x_\ell) = G_{x_\ell}(\cdots(G_{x_2}(G_{x_1}(k)))\cdots)$$

  where $G(\cdot) = G_0(\cdot)||G_1(\cdot)$

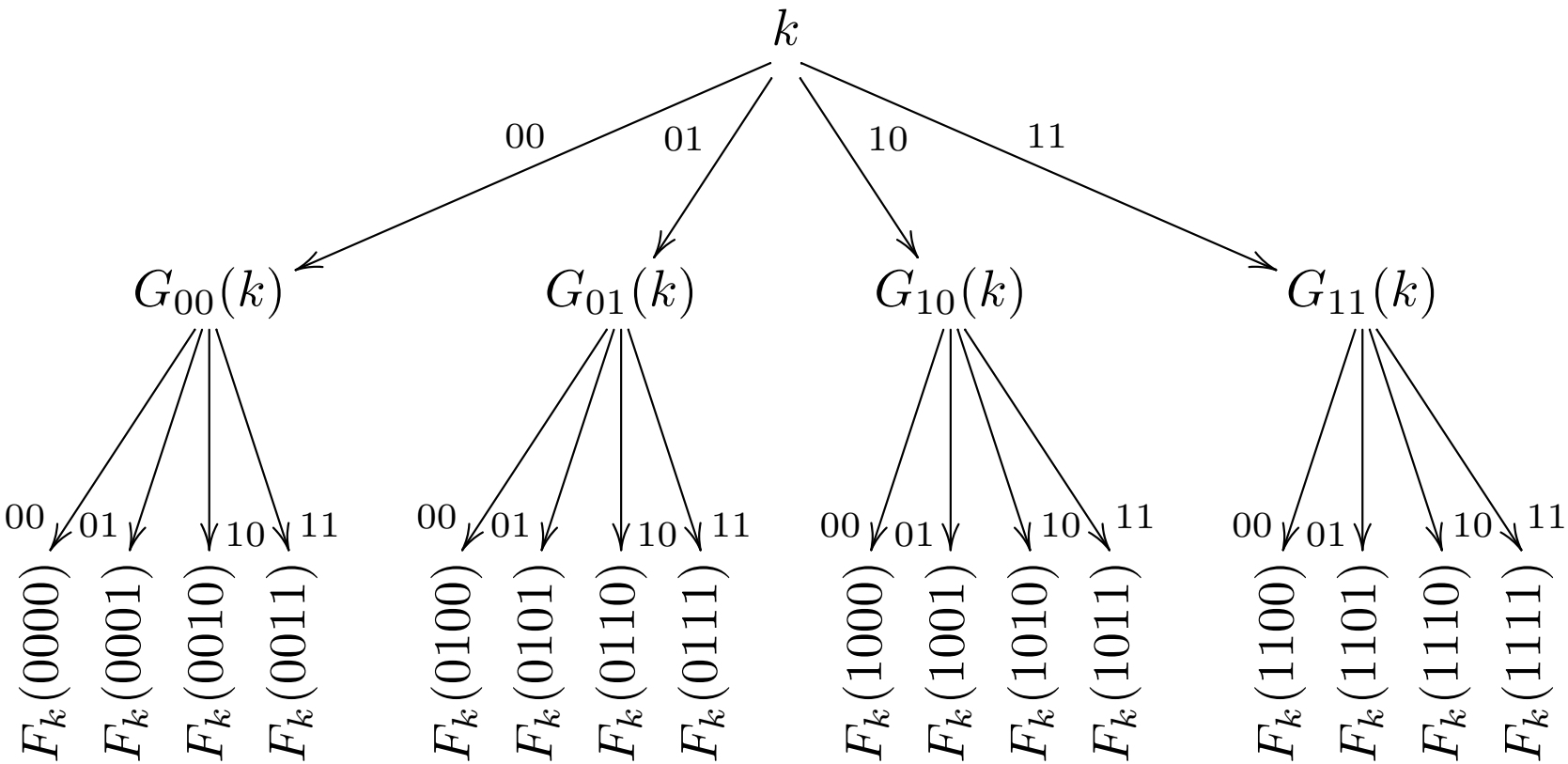
- can be depicted using binary tree

GGM-based PRF

Example: evaluate $F_k(01)$, i.e., $x = 01$

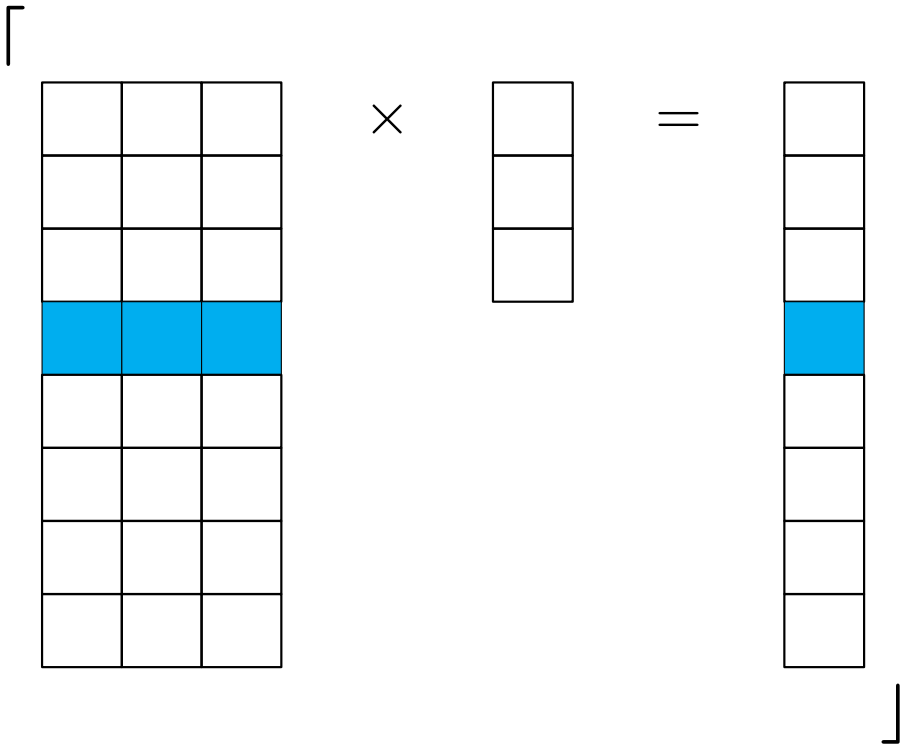


ω-ary tree



Module-Learning-with-Rounding

Informally,



$\sim$ uniformly random

Module-Learning-with-Rounding

More formally,

- let m and n be integer dimensions
- let moduli $q \geq p \geq 2$ be integers
- let $\mathcal{R} = \mathbb{Z}[x]/(x^d + 1)$ where d is a power of two
- let $\mathcal{R}_q = R/qR$

Module-Learning-with-Rounding

More formally,

- let m and n be integer dimensions
- let moduli $q \geq p \geq 2$ be integers
- let $\mathcal{R} = \mathbb{Z}[x]/(x^d + 1)$ where d is a power of two
- let $\mathcal{R}_q = R/qR$

The small-secret variant MLWR hardness assumption states that it is difficult to distinguish between the following two distributions:

$$(\mathbf{A}, \lfloor \mathbf{As} \rfloor_p) \text{ and } U(\mathcal{R}_q^{m \times n}) \times \lfloor U(\mathcal{R}_q^m) \rfloor_p$$

where $\mathbf{A} \sim U(\mathcal{R}_q^{m \times n})$ and $\mathbf{s} \sim U(\mathcal{R}_q^n)$

Our constructions

rely on

- GGM-based PRF using ω-ary tree
- MLWR-based PRG

Our constructions

rely on

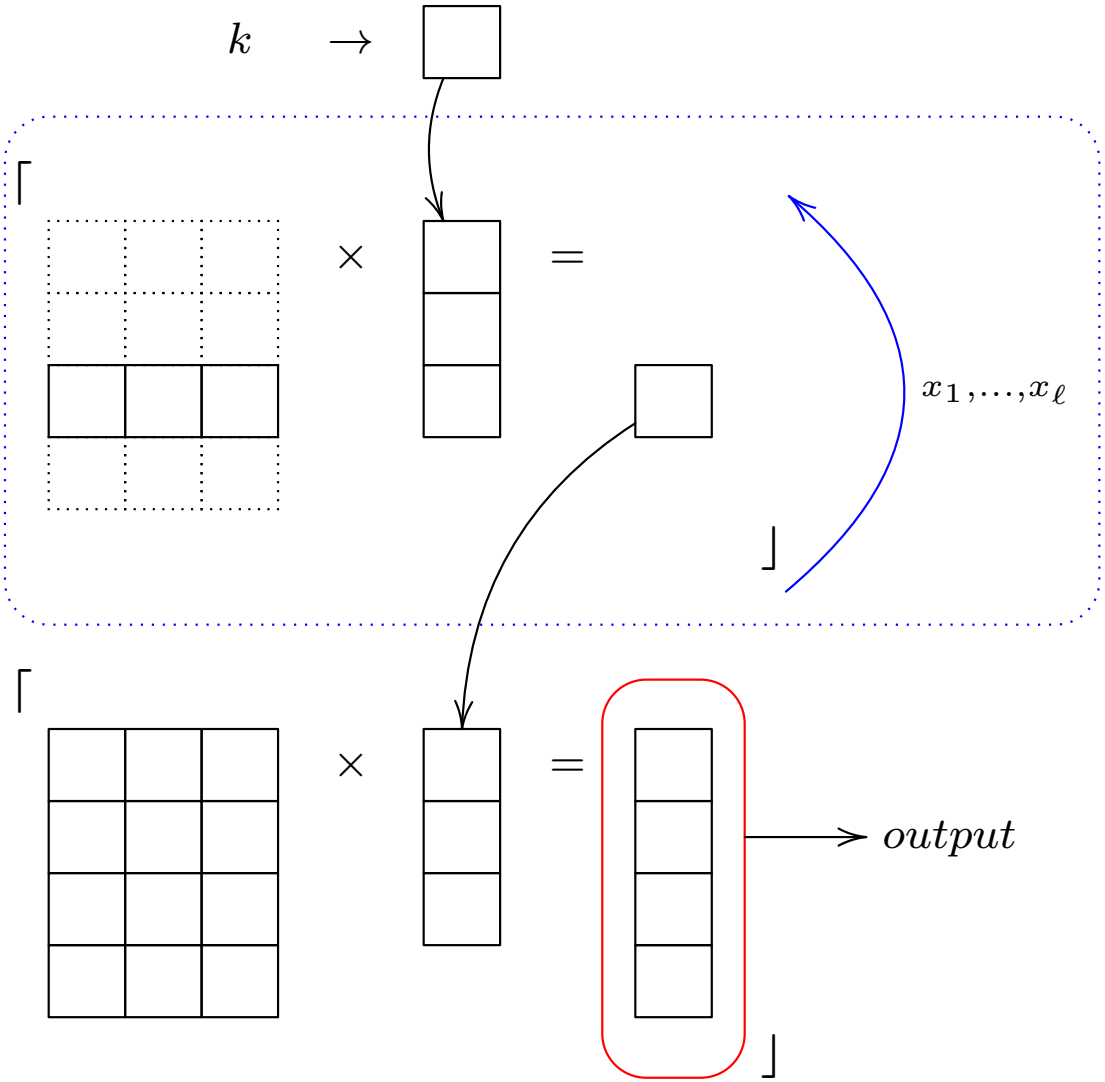
- GGM-based PRF using ω-ary tree
- MLWR-based PRG

differ from GGM84 in 3 ways

1. look at $\log \omega$ bits at a time
2. decrease tree's depth to $\ell / \log \omega$
3. need output to be ω longer

Outline

Input: PRF's key $k = \{0,1\}^m$ and a bit string $x = \{0,1\}^\ell$
Output: Pseudorandom string $z = \{0,1\}^{\ell'}$



Expansion rate

- Expansion rate $r =$ ratio between output and input length
- We want $r \geq \omega$
- Input length $s = n \cdot d \cdot (\log q - \log p)$
- Output length $\lfloor \mathbf{As} \rfloor_p = m \cdot d \cdot \log p$

$$\frac{m \cdot \quad d \quad \cdot \log p}{n \cdot \quad d \quad \cdot (\quad \log q - \log p \quad)} \geq r$$

Expansion rate

- Expansion rate $r =$ ratio between output and input length
- We want $r \geq \omega$
- Input length $s = n \cdot d \cdot (\log q - \log p)$
- Output length $\lfloor \mathbf{As} \rfloor_p = m \cdot d \cdot \log p$

$$\frac{m \cdot \quad d \quad \cdot \log p}{n \cdot \quad d \quad \cdot (\quad \log q - \log p \quad)} \geq r$$

example $$\frac{16 \cdot 256 \cdot \log 2^{12}}{3 \cdot 256 \cdot (\log 2^{16} - \log 2^{12})} \geq 16$$

Parameters

d	m	n	$\log B$	security	ω	mul	mem	depth	cost
128	4	5	4	145	2	69120	5120	128	8294400
128	4	6	2	131	4	41472	6144	64	2654208
128	16	5	4	133	8	69120	20480	43	2972160
128	32	5	4	133	16	69120	40960	32	2211840
256	4	3	4	167	4	62208	6144	64	3981312
256	8	3	4	167	8	62208	12288	43	2674944
256	16	3	4	167	16	62208	24576	32	1990656
256	32	3	4	167	32	62208	49152	26	1617408

- modulus $q = 2^{16}$
- $\log B = \log q - \log p$
- 'mem' : size in bytes of matrix $\mathbf{A}$
- 'depth' : tree's depth
- 'mul' : multiplication cost per depth
- 'cost' = mul $\times$ depth : total multiplication cost for full input length ℓ.

Parameters

d	m	n	$\log B$	security	ω	mul	mem	depth	cost
128	4	5	4	145	2	69120	5120	128	8294400
128	4	6	2	131	4	41472	6144	64	2654208
128	16	5	4	133	8	69120	20480	43	2972160
128	32	5	4	133	16	69120	40960	32	2211840
256	4	3	4	167	4	62208	6144	64	3981312
256	8	3	4	167	8	62208	12288	43	2674944
256	16	3	4	167	16	62208	24576	32	1990656
256	32	3	4	167	32	62208	49152	26	1617408

- modulus $q = 2^{16}$
- $\log B = \log q - \log p$
- 'mem' : size in bytes of matrix $\mathbf{A}$
- 'depth' : tree's depth
- 'mul' : multiplication cost per depth
- 'cost' = mul $\times$ depth : total multiplication cost for full input length ℓ.

Performance

Schemes	i7 Ivy Bridge	i5 Haswell	i7 Skylake
SPRING-BCH	46.0	19.5	-
SPRING-CRT	23.5	-	-
Ours, MLWR	-	-	39.4
Ours, LWR	-	-	64.2

- measured in terms of the number of cycles per output byte.

Note

- Haswell and Skylake have AVX2 instructions
- Highest CPU frequency of Haswell is higher than that of Skylake
  - Running ours on Haswell should not be slower than on Skylake

Summary

- Investigate efficiency of GGM using (M)LWR
- Propose lower-depth GGM-based PRF
- Bridge the gap between practical and provably secure PRFs
- Provide efficient implementations
- See paper: https://eprint.iacr.org/2021/446.pdf

Making it run really fast

- Maximize usage of available vector multipliers
- Eliminate redundant operations
  - precompute to reuse values
- Minimize overhead from permutations
  - organize data to fit instruction format
- Schedule instructions to keep CPU as busy as possible
- etc.