FOLEAGE: $\mathbb{F}_4$−OLE-Based MPC for Boolean Circuits

Maxime Bombar
maxime.bombar@math.u-bordeaux.fr
Université de Bordeaux (France)*

*Work done while Postdoc in CWI (Amsterdam, The Netherlands).

Dung Bui
Geoffroy Couteau
Clément Ducros
{bui, couteau, cducros}@irif.fr
IRIF (Paris, France)

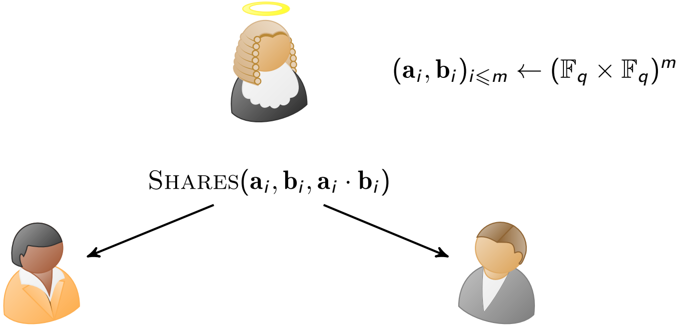
Alain Couvreur
alain.couvreur@inria.fr
Inria (France)

Sacha Servan-Schreiber
3s@mit.edu
MIT (USA)

Centrum Wiskunde & Informatica

MPC in the Correlated Randomness Model

Preprocessing

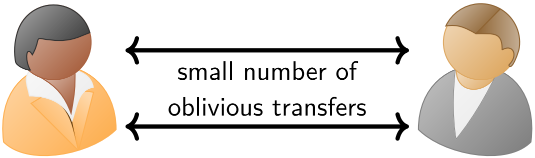


Online Phase

Fast online protocol using one $\mathbb{F}_q$-triple per multiplication

How to efficiently distribute m ($\approx 2^{30}$) random multiplication triples?

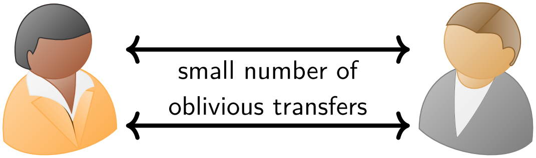
Traditional Approach: OT extensions

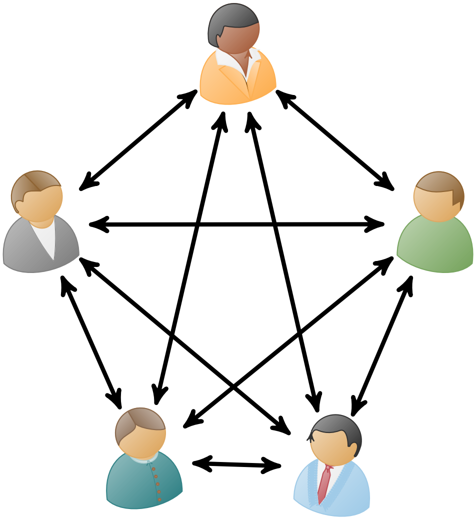


OT Extension (*e.g.* [IKNP03]):

Cheap symmetric cryptography to generate tons of OT.

Traditional Approach: OT extensions





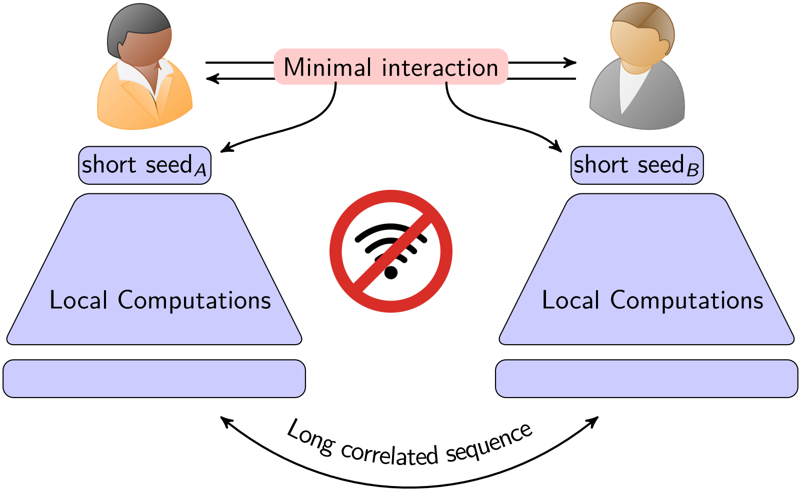
Communication scales as $\Omega(m \cdot N^2)$ for m triples. ✗

Practical Secure Computation over Large Fields

- SPDZ protocol leverages (somewhat) homomorphic encryption to scale as $O(m \cdot N)$.
- Overdrive [KPR18]: Good concrete efficiency ($\approx 10^5$ triples per second). ✓
- Only available over large fields. ✗

Damgård, Pastro, Smart, Zakarias - *MPC from somewhat homomorphic encryption* - CRYPTO 2012
Keller, Pastro, and Rotaru - *Overdrive: Making SPDZ great again* - EUROCRYPT 2018

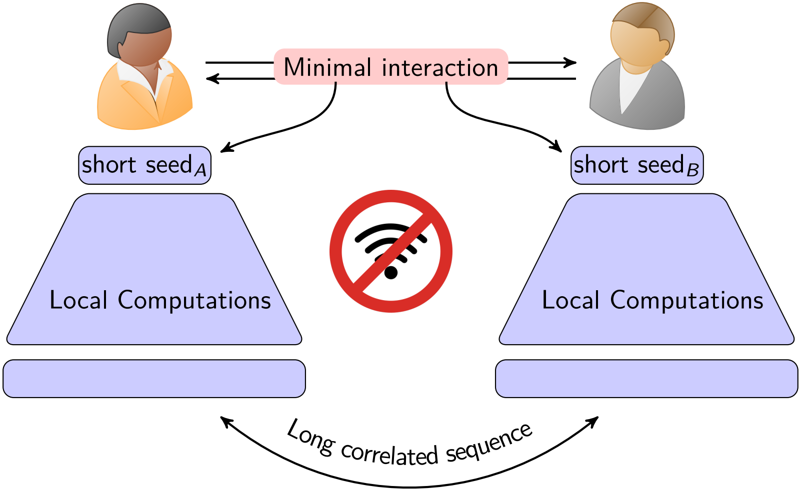
A New Tool: Programmable Pseudorandom Correlation Generators



Introduced by Boyle, Couteau, Gilboa, Ishai, Kohl, Sholl (2019, 2020)

Communication: $O(\log m \cdot N^2)$.
$\approx 10^5$ triples per second.

A New Tool: Programmable Pseudorandom Correlation Generators

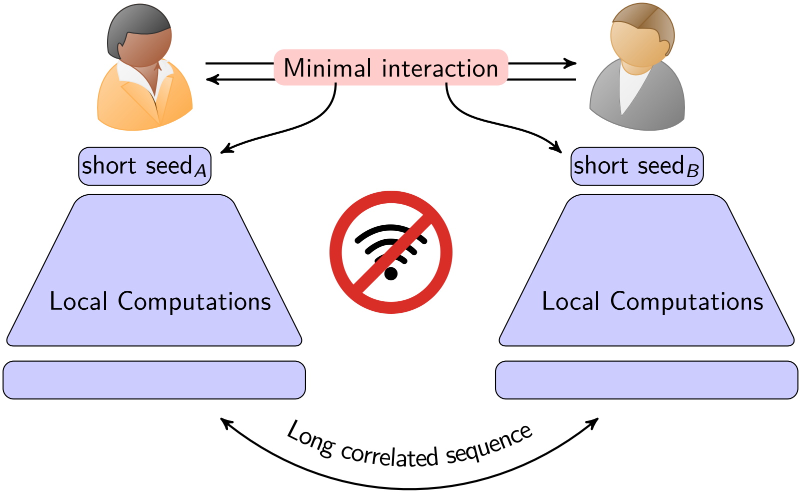


Introduced by Boyle, Couteau, Gilboa, Ishai, Kohl, Sholl (2019, 2020)

- Communication: $O(\log m \cdot N^2)$.
- $\approx 10^5$ triples per second.

N-party only possible over **large fields**.

A New Tool: ~~Programmable~~ Pseudorandom Correlation Generators



Introduced by Boyle, Couteau, Gilboa, Ishai, Kohl, Sholl (2019, 2020)

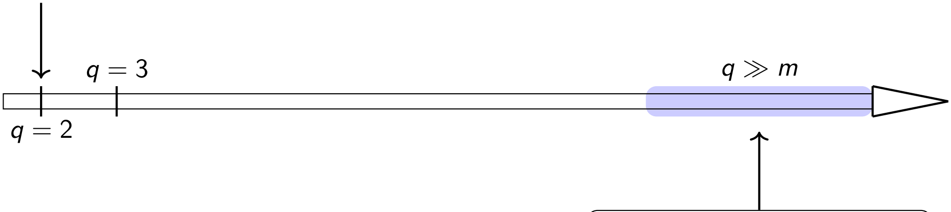
- Communication: $O(\log m)$.
- $\approx 10^6$ triples per second (SoftSpoken [Roy22], [RRT23])

Silent OT extensions.

L. Roy - *SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model* - CRYPTO 2022
Raghuraman, Rindal, Tanguy - *Expand-convolute codes for PCGs from LPN* - CRYPTO 2023

Landscape of Correlation Generators

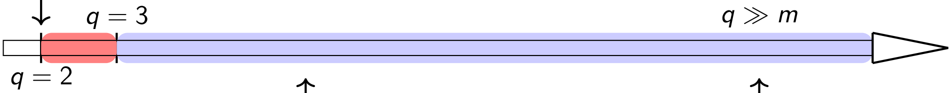
- [BCGIKS19, Roy22, RRT23]: $O(\log m)$ communication.
- $\approx 10^6$ triples per second. ✓
- Only for two parties. ✗



- [KPR18]: $O(m \cdot N)$.
- [BCGIKS20]: $O(\log m \cdot N^2)$
- $\approx 10^5$ triples per second.
- Large q. ✗

Landscape of Correlation Generators

- [BCGIKS19, Roy22, RRT23]: $O(\log m)$ communication.
- $\approx 10^6$ triples per second. ✓
- Only for two parties. ✗



Can we do better than $\Omega(N^2 m)$?

Practical efficiency?

- [**B**CCD23]: $O(\log m \cdot N^2)$
- $\approx 10^5$ triples per second (estimated).
- $q > 3$.

- [KPR18]: $O(m \cdot N)$.
- [BCGIKS20]: $O(\log m \cdot N^2)$
- $\approx 10^5$ triples per second.
- Large q. ✗

This Work: Best of Both Worlds

- **Silent preprocessing**: $O(\log m)$ communication
- $\approx 12.3 \cdot 10^6$ triples per second.
- **Small** seeds.

2-party

- **Almost silent preprocessing**: $O(\log m \cdot N^2) + m \cdot N$ communication.
- **Very low** computational overhead.
- **Parallelization** up to $2 \cdot (N-1)$ processors.
- **Faster** than Overdrive for $N \lessapprox 400$.
- **Optimizations** of independent interest.

N-party

- Novel protocol for computing **Boolean** circuits based on $\mathbb{F}_4$ precomputations, both for **two-party** and **N-party** settings.
- Low communication, low computational overhead.

$\mathbb{F}_4$oleage

Performance Comparison

	Communication	localhost	LAN	WAN
Multi-party setting ($N = 10$)				
SoftSpoken ($k = 2$)	134 GB	342s	1192s	12207s
SoftSpoken ($k = 4$)	67 GB	405s	596s	6104s
SoftSpoken ($k = 8$)	34 GB	1900s	**1900s**	3052s
			*298s	
RRT	6.3 GB	2619s	**2619s**	**2619s**
			*50.3s	*515s
$\mathbb{F}_4$OLEAGE	0.7 GB	1463s	**1463s**	**1463s**
			*5.6s	*57.9s
Two-party setting ($N = 2$)				
SoftSpoken ($k = 2$)	15 GB	38s	119s	1221s
SoftSpoken ($k = 4$)	7.5 GB	45s	60s	610s
SoftSpoken ($k = 8$)	3.7 GB	211s	**211s**	**211s**
RRT	258 KB	292s	**292s**	**292s**
$\mathbb{F}_4$OLEAGE	33.5 MB	81s	**81s**	**81s**

red: Bottleneck = local computations

A Framework for Programmable PCGs for $\mathbb{F}_q$-OLEs

Goal. Generate **a lot** of $\text{OLE}'s$ over $\mathbb{F}_q$.

Wishful thinking. Take a ring $\mathcal{R} \simeq \mathbb{F}_q \times \cdots \times \mathbb{F}_q$

⚠Not all rings $\mathcal{R}$ are secure.

ONE OLE over $\mathcal{R}$

$$\mathbf{U} \cdot \mathbf{V} = \mathbf{X} + \mathbf{Y}$$

Local Computations ↓

Many $\text{OLE}'s$ over $\mathbb{F}_q$

$$\mathbf{u}_i \cdot \mathbf{v}_i = \mathbf{x}_i + \mathbf{y}_i$$

Boyle, Couteau, Gilboa, Ishai, Kohl, and Sholl - *Efficient PCGs from Ring-LPN* - CRYPTO 2020
Boyle, Gilboa and Ishai - *Function secret sharing: Improvements and extensions* - CCS 2016

A Framework for Programmable PCGs for $\mathbb{F}_q$-OLEs

Goal. Generate **a lot** of OLE$'s$ over $\mathbb{F}_q$.

Example: $\mathcal{R} = \mathbb{F}_q[X]/(F(X))$ with F split.

ONE OLE over $\mathcal{R}$

Local Computations

Many OLE$'s$ over $\mathbb{F}_q$

$\mathbf{e}$ and $\mathbf{f}$ are **sparse**

$$\mathbf{U} \stackrel{\text{def}}{=} \mathbf{a} \cdot \mathbf{e}_u + \mathbf{f}_u \approx \$$$

$$\mathbf{V} \stackrel{\text{def}}{=} \mathbf{a} \cdot \mathbf{e}_v + \mathbf{f}_v \approx \$$$

$$\mathbf{U} \cdot \mathbf{V} = \mathbf{X} + \mathbf{Y}$$

$$\mathbf{U} \cdot \mathbf{V} = \mathbf{a}^2 \mathbf{e}_u \mathbf{e}_v + \mathbf{f}_u \mathbf{f}_v + \mathbf{a}(\mathbf{e}_u \mathbf{f}_v + \mathbf{e}_v \mathbf{f}_u)$$

$$\mathbf{u}_i \cdot \mathbf{v}_i = \mathbf{x}_i + \mathbf{y}_i$$

Cross-products are *sparse-ish* $\rightarrow$ Sharing via sums of *Distributed Point Functions* (DPF) [BGI16]

Boyle, Couteau, Gilboa, Ishai, Kohl, and Sholl - *Efficient PCGs from Ring-LPN* - CRYPTO 2020
Boyle, Gilboa and Ishai - *Function secret sharing: Improvements and extensions* - CCS 2016

Other Choice of Ring: Group Algebras

Finite abelian group G.

$$\mathbb{F}_q[G] = \left\{ \sum_{g \in G} a_g g \mid a_g \in \mathbb{F}_q \right\} \simeq \mathbb{F}_q^{|G|} \quad \text{also written as} \quad \left\{ \sum_{g \in G} a_g X^g \mid a_g \in \mathbb{F}_q \right\}$$

- $G = \{1\} \Rightarrow \mathbb{F}_q[G] = \mathbb{F}_q$.
- $G = \mathbb{Z}/n\mathbb{Z} \Rightarrow \mathbb{F}_q[G] = \mathbb{F}_q[X]/(X^n - 1)$
- $G = \mathbb{Z}/d_1\mathbb{Z} \times \cdots \times \mathbb{Z}/d_t\mathbb{Z} \Rightarrow \mathbb{F}_q[G] = \mathbb{F}_q[X_1, \ldots, X_t]/(X_1^{d_1} - 1, \ldots, X_t^{d_t} - 1)$
- $G = (\mathbb{Z}/(q-1)\mathbb{Z})^t = \mathbb{F}_q[X_1, \ldots, X_t]/(X_i^{q-1} - 1) \simeq \mathbb{F}_q^{(q-1)^t} \Longrightarrow$ Key to work over small fields.

Pseudorandomness of the OLE: **Quasi-Abelian Syndrome Decoding** assumption.

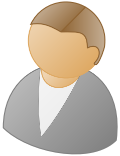
B., Couteau, Couvreur, Ducros - *PCGs from the Hardness of Quasi-Abelian Decoding* - CRYPTO 2023

A Programmable PCG for OLE over $\mathbb{F}_4$

Set $G = (\mathbb{Z}/3\mathbb{Z})^t$ and $\mathcal{R} = \mathbb{F}_4[G] = \mathbb{F}_4[X_1, \ldots, X_t]/(X_i^3 - 1)$

Seed Generation: Sample random **sparse** $\mathbf{e}_i, \mathbf{f}_j$ from $\mathcal{R} = \mathbb{F}_4[G]$.
Compute $\textsc{Shares}(\mathbf{e}_i\mathbf{f}_j)$ via *Function Secret Sharing*.

Distributed Setup: Doerner & shelat protocol.



$\textsc{Seed}_A = (\mathbf{a}, \mathbf{e}_u, \mathbf{f}_u, \textsc{Shares}(\mathbf{e}_i\mathbf{f}_j))$

$\downarrow$

Locally compute $\mathbf{U} = \mathbf{a}\mathbf{e}_u + \mathbf{f}_u$ and $\textsc{Share}(\mathbf{UV})$
$\Rightarrow$ OLE's over $\mathbb{F}_4$ via evaluation over $(\mathbb{F}_4^\times)^t$.

$\textsc{Seed}_B = (\mathbf{a}, \mathbf{e}_v, \mathbf{f}_v, \textsc{Shares}(\mathbf{e}_i\mathbf{f}_j))$

$\downarrow$

Locally compute $\mathbf{V} = \mathbf{a}\mathbf{e}_v + \mathbf{f}_v$ and $\textsc{Share}(\mathbf{UV})$
$\Rightarrow$ OLE's over $\mathbb{F}_4$ via evaluation over $(\mathbb{F}_4^\times)^t$.

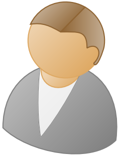
This paper: Blazingly fast implementation with **FFT** in $\mathbb{F}_4[G]$!

Let $([\![\mathbf{a}]\!], [\![\mathbf{b}]\!], [\![\mathbf{a} \cdot \mathbf{b}]\!])$ be an $\mathbb{F}_4$-Beaver triple.

$$\begin{aligned} [\![\mathbf{a}]\!] &= [\![\mathbf{a}_0]\!] + \theta \cdot [\![\mathbf{a}_1]\!] \\ [\![\mathbf{b}]\!] &= [\![\mathbf{b}_0]\!] + \theta \cdot [\![\mathbf{b}_1]\!] \\ [\![\mathbf{a} \cdot \mathbf{b}]\!] &= [\![\mathbf{c}_0]\!] + \theta \cdot [\![\mathbf{c}_1]\!] \end{aligned}$$

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= (\mathbf{a}_0\mathbf{b}_0 + \mathbf{a}_1\mathbf{b}_1) + \theta \cdot (\mathbf{a}_0\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_0 + \mathbf{a}_1\mathbf{b}_1) \\ &= \mathbf{c}_0 + \theta \mathbf{c}_1 \end{aligned}$$

$$\mathbf{a}_0\mathbf{b}_0 = \mathbf{c}_0 + \mathbf{a}_1\mathbf{b}_1$$



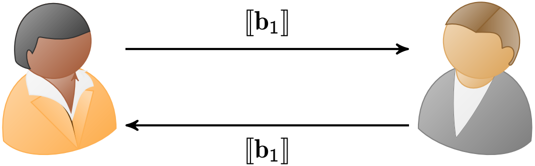
Optimization: Almost-Silent N-party Computation of Boolean Circuits

Let $([\![\mathbf{a}]\!], [\![\mathbf{b}]\!], [\![\mathbf{a}\cdot\mathbf{b}]\!])$ be an $\mathbb{F}_4$-Beaver triple.

$$\begin{aligned}
[\![\mathbf{a}]\!] &= [\![\mathbf{a}_0]\!] + \theta \cdot [\![\mathbf{a}_1]\!] \\
[\![\mathbf{b}]\!] &= [\![\mathbf{b}_0]\!] + \theta \cdot [\![\mathbf{b}_1]\!] \\
[\![\mathbf{a}\cdot\mathbf{b}]\!] &= [\![\mathbf{c}_0]\!] + \theta \cdot [\![\mathbf{c}_1]\!]
\end{aligned}$$

$$\begin{aligned}
\mathbf{a}\cdot\mathbf{b} &= (\mathbf{a}_0\mathbf{b}_0 + \mathbf{a}_1\mathbf{b}_1) + \theta\cdot(\mathbf{a}_0\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_0 + \mathbf{a}_1\mathbf{b}_1) \\
&= \mathbf{c}_0 + \theta\mathbf{c}_1
\end{aligned}$$

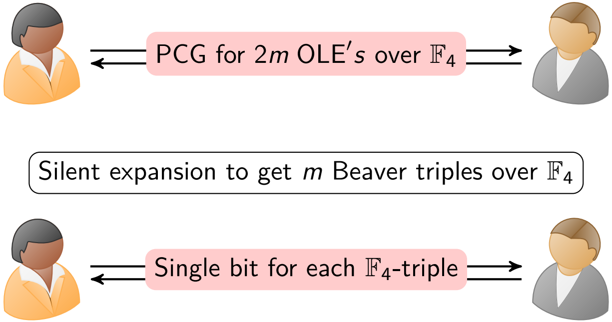
$$\mathbf{a}_0\mathbf{b}_0 = \mathbf{c}_0 + \mathbf{a}_1\mathbf{b}_1$$



$([\![\mathbf{a}_0]\!], [\![\mathbf{a}_1]\!], [\![\mathbf{c}_0]\!] + \mathbf{b}_1[\![\mathbf{a}_1]\!])$
is a **valid** $\mathbb{F}_2$-Beaver triple!

Single bit of communication per party.

Preprocessing



$O(\log(m))$ for each pair of parties.

Each party broadcasts a single bit per triple.

Local computation of m $\mathbb{F}_2$-triples

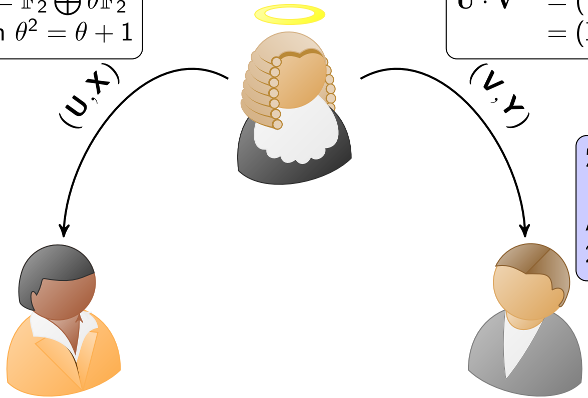
Online Phase

Fast online protocol using one $\mathbb{F}_2$-triple per AND gate

Improvement for 2-Party Computation

$\mathbb{F}_4 = \mathbb{F}_2 \bigoplus \theta \mathbb{F}_2$
with $\theta^2 = \theta + 1$

$$\begin{aligned} \mathbf{U} \cdot \mathbf{V} &= (\mathbf{U}_0\mathbf{V}_0 + \mathbf{U}_1\mathbf{V}_1) + \theta \cdot (\mathbf{U}_0\mathbf{V}_1 + \mathbf{U}_1\mathbf{V}_0 + \mathbf{U}_1\mathbf{V}_1) \\ &= (\mathbf{X}_0 + \mathbf{Y}_0) + \theta \cdot (\mathbf{X}_1 + \mathbf{Y}_1) \in \mathbb{F}_4 \end{aligned}$$



$\mathbf{Z}_A + \mathbf{Z}_B = (\mathbf{U}_0 + \mathbf{V}_1) \cdot (\mathbf{U}_1 + \mathbf{V}_0) \in \mathbb{F}_2$

A **single** $\mathbb{F}_4$-OLE yields one $\mathbb{F}_2$-Beaver triple.
2-party $\mathbb{F}_2$ Beaver triple is **silent**!

Doesn't extend to more than 2-parties

$$\mathbf{U} = \mathbf{U}_0 + \theta \cdot \mathbf{U}_1$$
$$\mathbf{X} = \mathbf{X}_0 + \theta \cdot \mathbf{X}_1$$

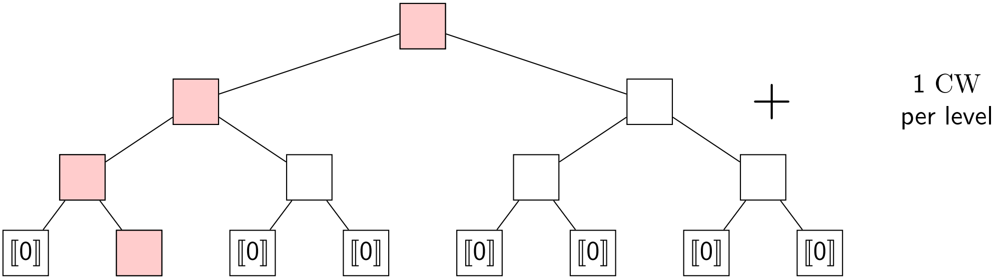
$$\mathbf{V} = \mathbf{V}_0 + \theta \cdot \mathbf{V}_1$$
$$\mathbf{X} = \mathbf{Y}_0 + \theta \cdot \mathbf{Y}_1$$

$\mathbf{Z}_A \overset{\text{def}}{=} \mathbf{U}_0\mathbf{U}_1 + \mathbf{X}_0 \in \mathbb{F}_2$

$\mathbf{Z}_B \overset{\text{def}}{=} \mathbf{V}_0\mathbf{V}_1 + \mathbf{Y}_0 \in \mathbb{F}_2$

PCG Seed Generation: 2-party DPF ([BGI15, BGI16, Ds17])

- GGM trees representing shares of a unit vector. Consistency is ensured by using (public) Correction Words.
- **Doerner and shelat protocol**: Distributed generation when the parties hold a **binary additive sharing** of the special path.
- Extension to t-sparse vectors: t-fold repetition and sum the point functions.



Optimization of the PCG Distributed Seed Generation

We need to create DPF for product of sparse elements $\mathbf{e}_i \cdot \mathbf{e}_j \in \mathbb{F}_4[X_1, \ldots, X_t]/(X_1^3 - 1, \ldots, X_t^3 - 1)$.

A monomial in $e_i \cdot e_j$ is of the form $\mathbf{X}^{p_i} \cdot \mathbf{X}^{p_j} = \mathbf{X}^{p_i + p_j \mod 3}$ where $p_i, p_j \in (\mathbb{Z}/3\mathbb{Z})^n$ held by different parties: $\implies$ the parties natively hold **ternary shares** of the noisy positions!

Previous constructions would run an additional protocol to turn it into binary additive sharing.

Sharing Vectors over $\mathbb{F}_3$: Ternary DPF

- We adapt the DPF construction with using **ternary** trees.
- Adaptation of Doerner-shelat, making use of $\binom{3}{1} - \mathrm{OT}$ and 3 CW per level.

- Native ternary sharing of the error positions $\Longrightarrow$ **saves half** the total number of OT and rounds.
- Expansion of the seed becomes 20% faster because of flatter tree.
- Number of rounds reduced from $\log_2(\frac{|G|}{t})$ to $\log_3(\frac{|G|}{t})$.

- Uses $\binom{3}{1} - \mathrm{OT}$ instead of $\binom{2}{1} - \mathrm{OT}$.
- PCG seed size $1.5\times$ larger.

PCG Evaluation Optimization

FFT in $\mathbb{F}_4[G]$ is **extremely fast**. The bottleneck in seed expansion is the evaluation of $(c \cdot t)^2$ DPFs.

We can benefit from standard optimizations of DPF:

- **Regular noise**: Error vectors split into t unit vectors of length $\frac{|G|}{t}$
  $\Longrightarrow$ **reduces evaluation domain**.
- **Early termination** technique [BGI16] for **small** output domain ($\mathbb{F}_4$ vs $\lambda-$bit field):
  $\Longrightarrow$ $64\times$ speedup!

FOLEAGE in short:

- Very efficient for large Boolean circuits, and up to $N \approx 400$ parties.
- Several layers of optimizations: algorithmic, protocol and implementation.
- Script for selecting QASD parameters.

Questions: Improving the ternary DPF? Truly efficient silent precomputation for Boolean circuits?

`https://github.com/sachaservan/FOLEAGE-PCG` `https://ia.cr/2024/429`

Thank You!

The Security Assumption: Quasi-Abelian Syndrome Decoding

The QASD assumption: Given a target weight t, and a compression factor c, it should hold that

$$((\mathbf{a}_1, \ldots, \mathbf{a}_{c-1}), \sum_{i=1}^{c-1} \mathbf{a}_i \mathbf{e}_i + \mathbf{e}_0) \approx ((\mathbf{a}_1, \ldots, \mathbf{a}_{c-1}), \mathbf{u}^{\text{unif}})$$

where $\mathbf{a}_i, \mathbf{u} \leftarrow \mathbb{F}_q[G]$ and $\mathbf{e}_j$ are c random t-sparse elements of $\mathbb{F}_q[G]$.

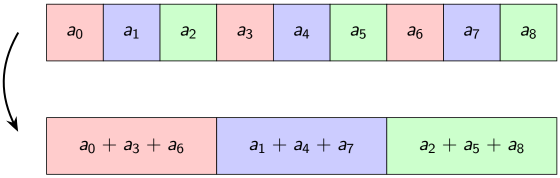
- Good minimum distance* $\Rightarrow$ resistance to all attacks from the linear test framework.
- Search-to-Decision reduction [**B**CCD23] and the search variant has been studied in algebraic coding theory for 50 years.

B., Couteau, Couvreur, Ducros - *PCG from the Hardness of Quasi-Abelian Decoding* - CRYPTO 2023

Concrete Analysis: Folding Attacks

Code-based analogue corresponds to **Folding attacks**, with respect to a subgroup H of G.

$$\pi_H : \begin{cases} \mathbb{F}_q[G] & \longrightarrow & \mathbb{F}_q[G/H] \\ \sum_{g \in G} a_g g & \longmapsto & \sum_{\bar{g} \in G/H} \left(\sum_{h \in H} a_{g+h} \right) \bar{g}. \end{cases}$$



- Fold along random subgroups until we get an easy instance (exponentially small probability).
- **This paper:** Precise **analysis** of these attacks and provides a script to determine secure parameters.