Correlated Pseudorandomness

Achieving faster secure computation through pseudorandom correlation generators

Geoffroy Couteau
Elette, Niv, and Yuval had just published ‘Breaking the Circuit Size Barrier for Secure Computation under DDH’ at CRYPTO’16
Elette, Niv, and Yuval had just published ‘Breaking the Circuit Size Barrier for Secure Computation under DDH’ at CRYPTO’16

I was at the time a young 2nd-year PhD student
Elette, Niv, and Yuval had just published ‘Breaking the Circuit Size Barrier for Secure Computation under DDH’ at CRYPTO’16.

I was at the time a young 2nd-year PhD student.

I got the feeling that this had to be useful to generate correlated randomness…
Elette, Niv, and Yuval had just published ‘Breaking the Circuit Size Barrier for Secure Computation under DDH’ at CRYPTO’16.

I was at the time a young 2nd-year PhD student. I got the feeling that this had to be useful to generate correlated randomness…

A few months and a CCS’17 paper later, we concluded that the answer was ‘not so much’.
7 years later
The Journey through Correlated Pseudorandomness
LPN is not LWE’s poor little brother
LPN is not LWE’s poor little brother

Deep connections to learning theory
The Journey through Correlated Pseudorandomness

- Deep connections to learning theory
- LPN is not LWE’s poor little brother
- Strong links with algebraic coding theory
- The Journey through Correlated Pseudorandomness
The Journey through Correlated Pseudorandomness

- LPN is not LWE’s poor little brother
- Deep connections to learning theory
- Strong links with algebraic coding theory
- Large-scale MPC on the internet is a possibility!
The Journey through Correlated Pseudorandomness

- LPN is not LWE’s poor little brother
- Deep connections to learning theory
- Strong links with algebraic coding theory
- Large-scale MPC on the internet is a possibility!
- Fast MPC + fun research
A Standard Cryptographic Approach

Idealized, information-theoretic object

Cryptographic compiler

Concrete crypto primitive
A Standard Cryptographic Approach

IOP, LPCP

SNARGS

Idealized, information-theoretic object

Cryptographic compiler

Concrete crypto primitive
A Standard Cryptographic Approach

Idealized, information-theoretic object → Cryptographic compiler → Concrete crypto primitive

IOP, LPCP

HBM, NIZKS

SNARKS
A Standard Cryptographic Approach

- Idealized, information-theoretic object
- Cryptographic compiler
- Concrete cryptographic primitive
- IOP, SNARGs, LPCP, HBM, NIZKS, OTP, Encryption
A Standard Cryptographic Approach

Idealized, information-theoretic object → Cryptographic compiler → Concrete crypto primitive

IOP, SNARGS, LPCP

HBM, NIZKS

Correlated randomness model

MPC

OTP

Encryption

⋯
A Standard Cryptographic Approach

Idealized, information-theoretic object → Cryptographic compiler → Concrete crypto primitive

IOP, SNARGs

HBM, NIZKS

Correlated randomness model

MPC

OTP

Encryption

⋯
Secure Computation…

- **Goal.** Computing a **public** function on secret inputs
- **Model.** \( n \) players, each with a private input \( x_i \) interacting through authenticated channels

\[
f : (x, y) \mapsto (z_A, z_B)
\]

- **Output:** Alice learns \( z_A \) and Bob learns \( z_B \)
- **Security:** Alice and Bob learn nothing else
... Is a Practical Concern.

- Use a dating app
- Search over our Cloud storage
- Get a recommendation on a streaming platform
- See a targeted advertising
- Use a social network
- Use a healthcare app
The Correlated Randomness Model

MPC protocol
The Correlated Randomness Model

$(r_1, r_2, r_3, r_4) \leftarrow \mathcal{D}$

MPC protocol
The Correlated Randomness Model

\( (r_1, r_2, r_3, r_4) \leftarrow \mathcal{D} \)

MPC protocol
The Correlated Randomness Model

Additive correlations

\[
\begin{align*}
    \forall (r_1, r_2, r_3, r_4) & \sim \text{shares}(r)
\end{align*}
\]

MPC protocol
Example: Beaver triples

$$(a_i, b_i)_{i \leq n} \leftarrow (\mathbb{F}_2 \times \mathbb{F}_2)^n$$

shares$$( (a_i, b_i, a_i \cdot b_i)_{i \leq n})$$

$2N$ bits / $\land$ gate for $N$ parties

GMW protocol
Example: Beaver triples

\[
(a_i, b_i)_{i \leq n} \leftarrow (\mathbb{F}_2 \times \mathbb{F}_2)^n
\]

shares\(((a_i, b_i, a_i \cdot b_i)_{i \leq n})\)

\[
2N \text{ bits} \wedge \text{ gate for } N \text{ parties}
\]
A Template to Instantiate *Efficiently* the Correlated Randomness Model

Given a correlation $C$, the dealer distributes shares of $C(r)$

Distributing $\langle C(r) \rangle$ *succinctly*
A Template to Instantiate *Efficiently* the Correlated Randomness Model

Given a correlation $C$, the dealer distributes shares of $C(r)$

Distributing $\langle C(r) \rangle$ *succinctly*

*Pseudorandom correlation generator*

Gen$(1^\lambda) \rightarrow (\text{seed}_A, \text{seed}_B)$ such that

1. $(\text{Expand}(A, \text{seed}_A), \text{Expand}(B, \text{seed}_B))$ looks like $n$ samples from the target correlation, and
2. $\text{Expand}(A, \text{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\text{Expand}(B, \text{seed}_B)$’ to Bob (similar property w.r.t. Alice).
A Template to Instantiate *Efficiently* the Correlated Randomness Model: MPC with silent preprocessing

**Pseudorandom correlation generator**: Gen$\left(1^k\right) \rightarrow (\text{seed}_A, \text{seed}_B)$ such that (1) $(\text{Expand}(A, \text{seed}_A), \text{Expand}(B, \text{seed}_B))$ looks like $n$ samples from the target correlation, and (2) $\text{Expand}(A, \text{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\text{Expand}(B, \text{seed}_B)$’ to Bob (similar property w.r.t. Alice).
A Template to Instantiate *Efficiently* the Correlated Randomness Model: MPC with silent preprocessing

**Pseudorandom correlation generator**: $\text{Gen}(1^\lambda) \rightarrow (\text{seed}_A, \text{seed}_B)$ such that (1) $(\text{Expand}(A, \text{seed}_A), \text{Expand}(B, \text{seed}_B))$ looks like $n$ samples from the target correlation, and (2) $\text{Expand}(A, \text{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\text{Expand}(B, \text{seed}_B)$’ to Bob (similar property w.r.t. Alice).

**One-time short interaction**

Interactive protocol with short communication and computation; Alice and Bob store a small seed afterwards.
A Template to Instantiate *Efficiently* the Correlated Randomness Model: MPC with silent preprocessing

**Pseudorandom correlation generator:** $\text{Gen}(1^\lambda) \rightarrow (\text{seed}_A, \text{seed}_B)$ such that (1) $(\text{Expand}(A, \text{seed}_A), \text{Expand}(B, \text{seed}_B))$ looks like $n$ samples from the target correlation, and (2) $\text{Expand}(A, \text{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\text{Expand}(B, \text{seed}_B)$’ to Bob (similar property w.r.t. Alice).

**Preprocessing phase**

Interactive protocol with short communication and computation; Alice and Bob store a small seed afterwards.

The bulk of the preprocessing phase is offline: Alice and Bob stretch their seeds into large pseudorandom correlated strings.

**Online phase**

Interactive protocol with short communication and computation; Alice and Bob store a small seed afterwards.

**One-time short interaction**

**‘Silent’ computation**
A Template to Instantiate *Efficiently* the Correlated Randomness Model: MPC with silent preprocessing

**Pseudorandom correlation generator:** $\text{Gen}(1^\lambda) \rightarrow (\text{seed}_A, \text{seed}_B)$ such that (1) $(\text{Expand}(A, \text{seed}_A), \text{Expand}(B, \text{seed}_B))$ looks like $n$ samples from the target correlation, and (2) $\text{Expand}(A, \text{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\text{Expand}(B, \text{seed}_B)$’ to Bob (similar property w.r.t. Alice).

**Preprocessing phase**

Interactive protocol with short communication and computation; Alice and Bob store a small seed afterwards.

**Online phase**

Alice and Bob consume the preprocessing material in a fast, non-cryptographic online phase.

**One-time short interaction**

- $\text{Gen}(1^\lambda)$
- $\text{seed}_A \rightarrow \text{seed}_B$

**‘Silent’ computation**

- $\text{Expand}(\text{seed}_A)$
- $\text{Expand}(\text{seed}_B)$

The bulk of the preprocessing phase is offline: Alice and Bob stretch their seeds into large pseudorandom correlated strings.
Q: What Correlations $C$ do we Consider?
Q: What Correlations $C$ do we Consider?

R: It depends! What MPC Protocol do you Want?

One does not simply build some MPC protocol
Q: What Correlations $C$ do we Consider?

R: It depends! What MPC Protocol do you Want?

One does not simply build some MPC protocol.
Q: What Correlations $C$ do we Consider?

R: It depends! What MPC Protocol do you Want?

One does not simply build some MPC protocol.

Depending on the application, you’ll want:

VOLE, OT, OLE, bilinear correlations, Beaver triples, authenticated Beaver triples, daBits, circuit-dependent correlations, polynomial correlations, matrix triples, OTTT…
Given a correlation $C$, the dealer distributes shares of $C(r)$
A Template to Instantiate *Efficiently* the Correlated Randomness Model

Given a correlation $C$, the dealer distributes shares of $C(r)$

Distributing $\langle C(r) \rangle$ *succinctly*

**Diagram:**
- **Correlation (function)**: $\langle C(r) \rangle$
  - Shares
  - Random coin
- **Public function**: $\text{FSS}(C \circ \text{PRG(seed)})$
  - Function secret sharing
  - Short seed
A Template to Instantiate **Efficiently** the Correlated Randomness Model

Given a correlation $C$, the dealer distributes shares of $C(r)$

Distributing $\langle C(r) \rangle$ **succinctly**

- **Correlation (function)**
  - Shares
  - Random coin

- **Public function**
  - $FSS(C \circ PRG(seed))$
  - Function secret sharing
  - Short seed
Function Secret Sharing

\[ \text{FSS} \ \left\{ \begin{array}{l}
\text{Share}(f) \mapsto (\text{Red} \ \text{Blue}) \\
\text{Eval}(\text{Red}, x) + \text{Eval}($\text{Blue}$, x) = f(x)
\end{array} \right. \]
Function Secret Sharing

\[ \text{FSS} \left\{ \begin{array}{l}
\text{Share}(f) \mapsto (\text{Red}, \text{Red}) \\
\text{Eval}(\text{Red}, x) + \text{Eval}(\text{Red}, x) = f(x)
\end{array} \right. \]

**Low end**
- OWF
- IT
- Point functions
- Lin. comb.

**Mid end**
- DDH, DCR, class groups
- NC¹

**High end**
- iO, Multi-key threshold FHE
- All polytime functions
Function Secret Sharing

\[ \text{Share}(f') \mapsto (\text{red block}, \text{red block}) \]

\[ \text{Eval}(\text{red block}, x) + \text{Eval}(\text{red block}, x) = f(x) \]

- **Low end**
  - OWF
  - Point functions
  - IT
  - Lin. comb.

- **Mid end**
  - DDH, DCR, class groups
  - \( \text{NC}^1 \)

- **High end**
  - iO, Multi-key threshold FHE
  - All polytime functions
What Functions can be Shared?

Sharing an arbitrary function: $f$
What Functions can be Shared?

Sharing an arbitrary function:

\[ f(0), f(1), f(2), f(3), \ldots, f(N) \]
What Functions can be Shared?

Sharing an arbitrary function:
What Functions can be Shared?

Sharing an arbitrary function:

\[ f_0, f_1, f_2, \ldots, f_N \]

\[
\begin{align*}
  f &\quad \text{FSS . Share}(f) \\
  f(0) & \\
  f(1) & \\
  f(2) & \\
  \vdots & \\
  f(N) &
\end{align*}
\]
What Functions can be Shared?

Sharing an arbitrary function: \( f \)
What Functions can be Shared?

Sharing an arbitrary function:

FSS. Eval($f$)
What Functions can be Shared?

Sharing an arbitrary function: $f$

Share size is $N$

FSS . Eval($f$)
What Functions can be Shared?
What Functions can be Shared?

succinctly

Sharing the all zero function: 
\[ \forall x, f(x) = 0 \]
What Functions can be Shared?

succinctly

Sharing the all zero function:
\[
\forall x, f(x) = 0
\]
What Functions can be Shared?

**succinctly**

Sharing the all zero function:
\[ \forall x, f(x) = 0 \]
What Functions can be Shared?

succinctly

Sharing the all zero function:
\[ \forall x, f(x) = 0 \]

Identical long random strings
What Functions can be Shared?

Sharing the all zero function:
\[ \forall x, f(x) = 0 \]

We just need a PRG!
What Functions can be Shared?

succinctly

Sharing a point function

\[ f_{\alpha, \beta}(x \neq \alpha) = 0, \; f(\alpha) = \beta \]
What Functions can be Shared?

**succinctly**

Sharing a point function

\[ f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta \]

All zero function
What Functions can be Shared?

**succinctly**

**Sharing a point function**

\[ f_{\alpha, \beta}(x \neq \alpha) = 0, f(\alpha) = \beta \]

...
What Functions can be Shared?

succinctly

Sharing a point function
\[ f_{\alpha, \beta}(x \neq \alpha) = 0, f(\alpha) = \beta \]
What Functions can be Shared? succinctly

Sharing a point function
\[ f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta \]
What Functions can be Shared?

**succinctly**

Sharing a point function
\[ f_{\alpha, \beta}(x \neq \alpha) = 0, f(\alpha) = \beta \]

\[ \Delta = \oplus \text{PRG(seed}_{\alpha_1} \oplus \text{PRG(seed}_{\alpha_1}' \right) \]

\[
\begin{array}{c}
\text{seed}_i \\
\vdots \\
\text{seed}_{\alpha_1} \\
\vdots \\
\text{seed}_{\sqrt{N}} \\
\end{array}
\]

\[
\begin{array}{c}
\text{seed}_i \\
\vdots \\
\text{seed}_{\alpha_1} \\
\vdots \\
\text{seed}_{\sqrt{N}} \\
\end{array}
\]
What Functions can be Shared?

succinctly

Sharing a point function

\[ f_{\alpha, \beta}(x \neq \alpha) = 0, f(\alpha) = \beta \]

Public

\[ \Delta = \begin{array}{c}
\beta \\
\oplus PRG(seed_{\alpha_1}) \oplus PRG(seed'_{\alpha_1})
\end{array} \]

\[
\begin{align*}
&b_i \quad seed_i \\
&... \\
&b_{\alpha_1} \quad seed_{\alpha_1} \\
&... \\
&b_{\sqrt{N}} \quad seed_{\sqrt{N}} \\
&seed_{\alpha_1} \quad b_{\alpha_1} \\
&\vdots \\
&seed_{\sqrt{N}} \quad b_{\sqrt{N}}
\end{align*}
\]
What Functions can be Shared?

succinctly

Sharing a point function
\[ f_{\alpha, \beta}(x \neq \alpha) = 0, f(\alpha) = \beta \]

Public

\[ \Delta = \beta + \bigoplus \text{PRG}(\text{seed}_{\alpha_1}) \bigoplus \text{PRG}(\text{seed}'_{\alpha_1}) \]

Eval:

\[ \text{PRG}(\text{seed}_j) \bigoplus \Delta \cdot b_j \]
What Functions can be Shared?

Sharing a point function
\[ f_{\alpha, \beta}(x \neq \alpha) = 0, \quad f(\alpha) = \beta \]

Recurring:

\[ b_i \quad \text{seed}_i \]
\[ \vdots \]
\[ b_{\alpha_1} \quad \text{seed}_{\alpha_1} \]
\[ \vdots \]
\[ b_{\sqrt{N}} \quad \text{seed}_{\sqrt{N}} \]

Giving \( \Delta \) and sharing the \( b_i \)'s are both essentially sharing a \( \sqrt{N} \)-size point function again: we can recurse the process!

\[ \text{seed}_i \quad b_i \]
\[ \vdots \]
\[ \text{seed}_{\alpha_1} \quad 1 - b_{\alpha_1} \]
\[ \vdots \]
\[ \text{seed}_{\sqrt{N}} \quad b_{\sqrt{N}} \]
What Functions can be Shared?

**succinctly**

Sharing a point function

\[ f_{\alpha, \beta}(x \neq \alpha) = 0, f(\alpha) = \beta \]

Recursing:

\[
\begin{align*}
& b_i \quad \text{seed}_i \\
& \vdots \\
& b_{\sqrt{N}} \quad \text{seed}_{\sqrt{N}} \\
& b_{\alpha_1} \quad \text{seed}_{\alpha_1} \\
& \vdots \\
& b_\lambda \quad \text{seed}_\lambda
\end{align*}
\]

Giving \( \Delta \) and sharing the \( b_i \)'s are both essentially sharing a \( \sqrt{N} \)-size point function again: we can recurse the process!

This + later improvements [BGI16]:

FSS for point functions with keys of size

\[ O(\lambda \cdot \log N) \]
We can succinctly share point functions

Public function

\[ FSS(C \circ \text{PRG}(\text{seed})) \]

Function secret sharing  Short seed

We can succinctly share point functions
We can succinctly share point functions $\mathcal{F}(C \circ \text{PRG}(\text{seed}))$.

Function secret sharing

Short seed

Secret sharing is additively homomorphic!

Linear combinations of
We can succinctly share point functions

\[ \text{FSS}(C \cdot \text{PRG}(\text{seed})) \]

Function secret sharing  \hspace{1cm} Short seed

Are there any PRGs in this class?

Secret sharing is additively homomorphic!

Public function

Linear combinations of
LPN to the Rescue

The LPN assumption - primal

\[
\begin{pmatrix}
G \\
\end{pmatrix}
, \\
\begin{pmatrix}
G \\
\end{pmatrix}
+ \\
\approx$

- Random matrix
- Short secret
- Sparse noise
LPN to the Rescue

The LPN assumption - primal

$H \cdot (G \cdot G + \text{Sparse noise}) \approx$ $\cdot$
The LPN assumption - primal

\[ H \cdot \left( \begin{array}{c} C \\ C \end{array} \right) + \text{Sparse noise} \approx \$ \]

Parity-check matrix of \( G \)
Random matrix
Short secret
LPN to the Rescue

The LPN assumption - dual

\[ H \cdot H \approx $ \]

Random matrix

Sparse noise
LPN to the Rescue

The LPN assumption - dual

\[ H \cdot H \approx \$ \]

Random matrix \( H \)

Sparse noise \( H \)

LPN yields a simple PRG in the class:
LPN to the Rescue

The LPN assumption - dual

Random matrix \( H \)

\( H \) \cdot Sparse noise

\( \approx \$ \)

LPN yields a simple PRG in the class:

\[ \text{PRG} : (\alpha_i)_{i \leq t} \mapsto H \cdot \sum_{i=1}^{t} \vec{u}^{\alpha_i}, \text{where } \vec{u}^{\alpha_i} \text{ is the unit vector with a 1 at } \alpha_i \]
LPN to the Rescue

The LPN assumption - dual

Random matrix

Sparse noise

LPN yields a simple PRG in the class:

PRG : \((\alpha_i)_{i \leq t} \mapsto H \cdot \sum_{i=1}^{t} \vec{u}_{\alpha_i}\), where \(\vec{u}_{\alpha_i}\) is the unit vector with a 1 at \(\alpha_i\)

Linear combination
LPN to the Rescue

The LPN assumption - dual

Random matrix $H$, Sparse noise $H \cdot \approx$

LPN yields a simple PRG in the class:

PRG : $(\alpha_i)_{i \leq t} \mapsto H \cdot \sum_{i=1}^{t} \vec{u}_{\alpha_i}$, where $\vec{u}_{\alpha_i}$ is the unit vector with a 1 at $\alpha_i$

Linear combination

(Truth table of) point functions
We have FSS for a class that contains a PRG.

\[ \text{FSS}(C \circ \text{PRG}(\text{seed})) \]

Public function

Function secret sharing
Short seed
We have FSS for a class that contains a PRG

The heavy lifting in the many subsequent works boils down to:

• Making the PRG more efficient
• Adding support for more complex $C$
We have FSS for a class that contains a PRG

The heavy lifting in the many subsequent works boils down to:

- Making the PRG more efficient
- Adding support for more complex $C$

Both questions are deeply rooted in (combinatorial and algebraic) coding theory.
We have FSS for a class that contains a PRG

The heavy lifting in the many subsequent works boils down to:

• Making the PRG more efficient
• Adding support for more complex $C$

Both questions are deeply rooted in (combinatorial and algebraic) coding theory
LPN and LWE

Digression: LPN versus LWE

LPN\((\mathbb{F}_2)\):
\[
G \leftarrow \mathbb{F}_2^{m \times n}, \quad \mathcal{B} \leftarrow \mathbb{F}_2^n,
\]
Ber\((\mathbb{F}_2)^n\)

‘Sparse’

LWE\((\mathbb{F}_p)\):
\[
G \leftarrow \mathbb{F}_p^{m \times n}, \quad \mathcal{B} \leftarrow \mathbb{F}_p^n,
\]
\([-B, B]^n\)

‘Small’
Digression: LPN versus LWE

LPN($\mathbb{F}_2$): $H \leftarrow \mathbb{F}_2^{m \times n}$, \( \text{Ber}(\mathbb{F}_2)^n \) ‘Sparse’

LWE($\mathbb{F}_p$): $H \leftarrow \mathbb{F}_p^{m \times n}$, $[-B, B]^n$ ‘Small’
Digression: LPN versus LWE

**LPN and LWE**

\[
\begin{pmatrix}
H \\
\cdot
\end{pmatrix}
\approx$

Random matrix

Noise

**LPN(\mathbb{F}_2):**

\[H \leftarrow \mathbb{F}_2^{m \times n},\]

\[\text{Ber}(\mathbb{F}_2)^n,\]

‘Sparse’

**LWE(\mathbb{F}_p):**

\[H \leftarrow \mathbb{F}_p^{m \times n},\]

\[[-B, B]^n,\]

‘Small’

\(O(n)\) entropy in the noise \(\Rightarrow\) LHL, statistical security, lattice trapdoors, lossiness…

Statistical security
LPN and LWE

LPN($\mathbb{F}_2$):

\[
H \leftarrow \mathbb{F}_2^{m \times n},
\]

Random matrix

Ber($\mathbb{F}_2)^n$

‘Sparse’

\[t \cdot \log n \ll n\] entropy in the noise \(\implies\) compressibility! Crucially used in recent results: PCGs, but also iO and batch OT.

LWE($\mathbb{F}_p$):

\[
H \leftarrow \mathbb{F}_p^{m \times n},
\]

\([-B, B]^n\)

‘Small’

\[O(n)\] entropy in the noise \(\implies\) LHL, statistical security, lattice trapdoors, lossiness…

Digression: LPN versus LWE

\[\approx \]

Noise
Some of my Favourite Open Questions

- Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?
- Can we go below $s/\log \log s$?
- Can we build 3-party PCGs?
- How fast can PCGs be?
- The communication of MPC with correlated randomness
- Public key PCFs: MPC as easy as key exchange
- The quest for the perfect code
Some of my Favourite Open Questions

- Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?
- Can we go below $s/\log \log s$?
- Can we get fast, usable, scalable MPC over the internet?
- Can we build 3-party PCGs?
- How fast can PCGs be?
Some of my Favourite Open Questions

Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

Can we go below $s/\log \log s$?

Can we get fast, usable, scalable MPC over the internet?

The communication of MPC with correlated randomness

The $\mathbb{F}_2$ barrier

Public key PCFs: MPC as easy as key exchange

Some of my favourite open question

The quest for the perfect code

Can we build 3-party PCGs?

How fast can PCGs be?
Making the PRG more Efficient
Making the PRG more Efficient

PRG : \((\alpha_i)_{i \leq t} \mapsto H \cdot \left( \begin{array}{c} + \ \ + \ \ + \ \ + \end{array} \right)\)
Making the PRG more Efficient

PRG : \((\alpha_i)_{i \leq t} \mapsto H \cdot \begin{pmatrix} + & + & + \end{pmatrix}\)

Generating and summing unit vectors

Multiplying by a random matrix of size \(\Omega(n^2)\)

Generating and summing unit vectors
Making the PRG more Efficient

PRG : \((\alpha_i)_{i \leq t} \mapsto H \cdot (\begin{array}{c} + \\cdots \\cdots +\end{array})\)

- Multiplying by a random matrix of size \(\Omega(n^2)\)
- Generating and summing unit vectors

\(n\) is the total amount of correlated randomness we want to generate! (Think: \(n \sim 2^{30}\))
Making the PRG more Efficient

\[ \text{PRG} : (\alpha_i)_{i \leq t} \rightarrow H \cdot \left( \begin{array}{c} + \\ + \\ + \\ + \end{array} \right) \]

Multiplying by a random matrix of size \(\Omega(n^2)\)

Generating and summing unit vectors

\textbf{⚠️} \(n\) is the total amount of correlated randomness we want to generate! (Think: \(n \sim 2^{30}\))

\textbf{❓} Can we replace \(H\) with a matrix that allows for fast matrix-vector product?

We need a rule of thumb to know which matrices will yield \textit{plausible} variants of LPN
A tremendous number of attacks on LPN have been published…

**Crucial observation:** most attacks fit in the same framework, the *linear test framework.* (*)

---

**Game**

1. Send $H$ to $\mathcal{A}$

2. \( \mathcal{A} \) returns a test vector $\vec{v}$ computed from $H$ in unbounded time

---

**Check**

The adversary wins in the distribution induced by

\[
\begin{pmatrix}
H \\
\vec{v}
\end{pmatrix}
\]

(over a random choice of secret and sparse noise) is non-negligibly biased.

---

- **Gaussian Elimination attacks**
  - Standard gaussian elimination
  - Blum-Kalai-Wasserman [J.ACM:BKW03]
  - Sample-efficient BKW [A-R:Lyu05]
  - Pooled Gauss [CRYPTO:EK17]
  - Well-pooled Gauss [CRYPTO:EK17]
  - Levieil-Fouque [SCN:LF06]
  - Covering codes [JC:GJL19]
  - Covering codes+ [BTV15]
  - Covering codes++ [BV:AC16]
  - Covering codes+++ [EC:ZJW16]

- **Information Set Decoding Attacks**
  - Prange's algorithm [Prange62]
  - Stern's variant [ICIT:Stern88]
  - Finiasz and Sendrier's variant [AC:FS09]
  - BJMM variant [EC:BJMM12]
  - May-Ozerov variant [EC:MO15]
  - Both-May variant [PQC:BM18]
  - MMT variant [AC:MMT11]
  - Well-pooled MMT [CRYPTO:EK17]
  - BLP variant [CRYPTO:BLP11]

- **Statistical Decoding Attacks**
  - Jabri's attack [CCC:Jab01]
  - Overbeck's variant [ACISP:Ove06]
  - FKI's variant [Trans.1T;FK106]
  - Debris-Tillich variant [ISIT:DT17]

- **Other Attacks**
  - Generalized birthday [CRYPTO:Wag02]
  - Improved GBA [Kirchner11]
  - Linearization [EC:BM97]
  - Linearization 2 [NDO:Saa07]
  - Low-weight parity-check [Zichron17]
  - Low-deg approx [ITCS:ABGKR17]
A tremendous number of attacks on LPN have been published…

**Crucial observation:** most attacks fit in the same framework, the *linear test framework*. (*)

---

**Linear Test Framework**

(*): highly structured algebraic codes (e.g., Reed-Solomon) are a different beast

---

### Gaussian Elimination attacks
- Standard gaussian elimination
- Blum-Kalai-Wasserman [J.ACM:BKW03]
- Sample-efficient BKW [A-R:Lyu05]
- Pooled Gauss [CRYPTO:EKM17]
- Well-pooled Gauss [CRYPTO:EKM17]
- Levisel-Fouque [SCN:LF06]
- Covering codes [JC:GJL19]
- Covering codes+ [BTV15]
- Covering codes++ [BV:AC16]
- Covering codes+++ [EC:ZJW16]

### Statistical Decoding Attacks
- Jabri’s attack [CCC:Jab01]
- Overbeck’s variant [ACISP:Ove06]
- FKI’s variant [Trans.IT:FKI06]
- Debris-Tillich variant [ISIT:DT17]

### Information Set Decoding Attacks
- Prange’s algorithm [Prange62]
- Stern’s variant [ICIT:Stern88]
- Finiasz and Sendrier’s variant [AC:FS09]
- BJMM variant [EC:BJMM12]
- May-Ozerov variant [EC:MO15]
- Both-May variant [PQC:BM18]
- MMT variant [AC:MTT11]
- Well-pooled MMT [CRYPTO:EKM17]
- BLP variant [CRYPTO:BLP11]

### Other Attacks
- Generalized birthday [CRYPTO:Wag02]
- Improved GBA [Kirchner11]
- Linearization [EC:BM97]
- Linearization 2 [INDO:Saa07]
- Low-weight parity-check [Zichron17]
- Low-deg approx [ITCS:ABGKR17]

---

**Game**

1. Send $H$ to
2. returns a *test vector* $\vec{v}$ computed from $H$ in unbounded time

---

**Check**

The adversary wins in the distribution induced by

(over a random choice of secret and sparse noise) is non-negligibly biased.
The adversary wins in the distribution induced by

\[ \vec{v} \cdot (G \cdot \vec{s} + \vec{e}) \]

(over a random choice of secret and sparse noise) is non-negligibly biased.

Claim: Assume \( t \) (number of noisy coordinates) is set to a security parameter. If there is a constant \( c \) such that every subset of \( c \cdot n \) rows of \( G \) is linearly independent, no linear test can distinguish \( G \cdot \vec{s} + \vec{e} \) from random.

We have a sum of two distributions:

- Induced by the codeword
- Induced by the noise vector

Protects against light linear tests

Protects against heavy linear tests
The adversary wins in the distribution induced by

$\vec{v} \cdot (G \cdot \vec{s} + \vec{e})$

(over a random choice of secret and sparse noise) is non-negligibly biased.

**Claim:** Assume $t$ (number of noisy coordinates) is set to a security parameter. If there is a constant $c$ such that every subset of $c \cdot n$ rows of $G$ is linearly independent, no linear test can distinguish $G \cdot \vec{s} + \vec{e}$ from random.

**Rephrasing the sufficient condition:**

Every subset of $O(n)$ rows of $G$ is linearly independent

$\iff$ the left-kernel of $G$ does not contain nonzero vector of weight less than $O(n)$

$\iff$ the dual code of $G$, i.e., the code generated by the transpose of its parity check matrix $H$, has linear minimum distance

We have a sum of two distributions:

Induced by the codeword

Protects against \textit{light} linear tests

Induced by the noise vector

Protects against \textit{heavy} linear tests
Pseudorandom Correlation Generators - Efficiency

**Goal**: computing fast, such that the code is LPN-friendly

We want to find a matrix $M = H^\top$ such that (1) the code generated by $M$ is a good code, and (2) computing $M \cdot \vec{v}$ takes time $O(n)$ for any $\vec{v}$ (this is the transposition principle)

$\implies$ We need to find a *good* and *linear-time encodable* code. And we want it concretely efficient!
There is an ongoing and exciting quest for pinpointing the *right* code for PCG applications:

- **CCS:** Boyle-C-Gilboa-Ishai’18 suggested using LDPC code
- **CCS:** Boyle-C-Gilboa-Ishai-Kohl-Rindal-Scholl’19 moved to quasi-cyclic codes 
  *due to concern regarding linear-time encoding of LDPC codes*
- **Crypto:** C-Raghuraman-Rindal’21: tailored LDPC with heuristic & experimental support
- **Crypto:** Boyle-C-Gilboa-Ishai-Kohl-Resch—Scholl’22: Expand-Accumulate codes
- Latest news: there’s apparently a new proposal that suggests Expand-Convolute codes instead (and which breaks Silver along the way!)
- There are a few more codes I’d like to investigate, the quest continues!
Some of my Favourite Open Questions

- Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?
- Can we build 3-party PCGs?
- Can we go below $s/\log \log s$?
- Can we get fast, usable, scalable MPC over the internet?
- How fast can PCGs be?

- The $\mathbb{F}_2$ barrier
- The 2-party barrier
- Some of my favourite open question
- The quest for the perfect code
- Public key PCFs: MPC as easy as key exchange
- The communication of MPC with correlated randomness
Some of my Favourite Open Questions

- Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?
- Can we go below $s/\log \log s$?
- Can we build 3-party PCGs?
- How fast can PCGs be?
- Can we get fast, usable, scalable MPC over the internet?
- The communication of MPC with correlated randomness
- Public key PCFs: MPC as easy as key exchange
- The quest for the perfect code
- The $\mathbb{F}_2$ barrier
OLE Correlations

OLE over $\mathbb{F}$ is the type of correlation we want to do (semi-honest) secure computation of arithmetic circuits over $\mathbb{F}$.

In an OLE, Alice gets $a \leftarrow \mathbb{F}$, Bob gets $b \leftarrow \mathbb{F}$, and Alice and Bob get random shares of $a \cdot b$. 
OLE Correlations, the LPN Way

Goal:

- Alice gets a pseudorandom vector $\vec{x}$
- Bob gets a pseudorandom vector $\vec{y}$
- Alice and Bob get shares of $\vec{x} \otimes \vec{y}$
Goal:

- Alice gets a pseudorandom vector $\vec{x}$
- Bob gets a pseudorandom vector $\vec{y}$
- Alice and Bob get shares of $\vec{x} \odot \vec{y}$
Goal:

- Alice gets a pseudorandom vector $\vec{x} = H \cdot \vec{e}_x$
- Bob gets a pseudorandom vector $\vec{y} = H \cdot \vec{e}_y$
- Alice and Bob get shares of $\vec{x} \odot \vec{y}$

OLE Correlations, the LPN Way
OLE Correlations, the LPN Way

Goal:

- Alice gets a pseudorandom vector $\vec{x} = H \cdot \vec{e}_x$
- Bob gets a pseudorandom vector $\vec{y} = H \cdot \vec{e}_y$
- Alice and Bob get shares of $\vec{x} \odot \vec{y}$
Goal:

- Alice gets a pseudorandom vector $\vec{x} = H \cdot \vec{e}_x$
- Bob gets a pseudorandom vector $\vec{y} = H \cdot \vec{e}_y$
- Alice and Bob get shares of $\vec{x} \odot \vec{y}$

OLE Correlations, the LPN Way
OLE Correlations, the LPN Way

Goal:
- Alice gets a pseudorandom vector $\vec{x} = H \cdot \vec{e}_x$
- Bob gets a pseudorandom vector $\vec{y} = H \cdot \vec{e}_y$
- Alice and Bob get shares of $\vec{x} \odot \vec{y}$

This is a $t^2$-sparse matrix, i.e. a sum of $t^2$ point functions! $\implies$ can be generated with comm. $O(\lambda t^2 \log n)$
Goal:

- Alice gets a pseudorandom vector \( \vec{x} = H \cdot \vec{e}_x \)
- Bob gets a pseudorandom vector \( \vec{y} = H \cdot \vec{e}_y \)
- Alice and Bob get shares of \( \vec{x} \odot \vec{y} \)

This is a \( t^2 \)-sparse matrix, i.e. a sum of \( t^2 \) point functions! \( \implies \) can be generated with comm. \( O(\lambda t^2 \log n) \)

\[
\begin{align*}
\vec{x} \cdot \vec{y} \quad &= \quad H \cdot \vec{e}_x \cdot \vec{e}_y \\
\end{align*}
\]

Uses only LPN

Costs \( \omega(n^2)! \) (Think: \( n \sim 2^{30} \ldots \))
Let $\mathcal{R}$ be the ring $\mathbb{Z}_p / F(X)$ where $F(X)$ is a degree-$n$ polynomial that splits entirely, and $p > n$.

**Ring-LPN assumption:** $(a, b) \sim (a, a \cdot e + f)$ where $(a, b) \leftarrow \mathcal{R}$ and $(e, f)$ are random $t$-sparse polynomials.

**Observation:** we can get $n$ OLE correlations from a single ‘ring-OLE’ correlation $(x, y, \langle x \cdot y \rangle)$ over $\mathcal{R}$: the OLE correlations are obtained by reducing $x$, $y$, and $x \cdot y$ modulo each of the linear factors $F_i$ of $F$.

**Construction:**
- Alice gets a pseudorandom polynomial $x = a \cdot e_x + f_x$ where $(e_x, f_x)$ are $t$-sparse polynomials over $\mathcal{R}$
- Bob gets a pseudorandom vector $y = a \cdot e_y + f_y$ where $(e_y, f_y)$ are $t$-sparse polynomials over $\mathcal{R}$
- Alice and Bob get shares of $x \cdot y = a^2 \cdot (e_x e_y) + a \cdot (e_x f_y + f_x e_y) + f_x f_y$

The polynomials $a^2, a$ are public, and $e_x e_y, e_x f_y, f_x e_y, f_x f_y$ are all $t^2$-sparse polynomials.
Let $\mathcal{R}$ be the ring $\mathbb{Z}_p/F(X)$ where $F(X)$ is a degree-$n$ polynomial that splits entirely, and $p > n$.

**Ring-LPN assumption:** $(a, b) \sim (a, a \cdot e + f)$ where $(a, b) \leftarrow \mathcal{R}$ and $(e, f)$ are random $t$-sparse polynomials.

**Observation:** we can get $n$ OLE correlations from a single ‘ring-OLE’ correlation $(x, y, \langle x \cdot y \rangle)$ over $\mathcal{R}$: the OLE correlations are obtained by reducing $x$, $y$, and $x \cdot y$ modulo each of the linear factors $F_i$ of $F$.

**Construction:**
- Alice gets a pseudorandom polynomial $x = a \cdot e_x + f_x$ where $(e_x, f_x)$ are $t$-sparse polynomials over $\mathcal{R}$
- Bob gets a pseudorandom vector $y = a \cdot e_y + f_y$ where $(e_y, f_y)$ are $t$-sparse polynomials over $\mathcal{R}$
- Alice and Bob get shares of $x \cdot y = a^2 \cdot (e_x e_y) + a \cdot (e_x f_y + f_x e_y) + f_x f_y$

The polynomials $a^2$, $a$ are public, and $e_x e_y, e_x f_y, f_x e_y, f_x f_y$ are all $t^2$-sparse polynomials.

\[\text{Costs only } O(n \cdot \log n)\]
Let $\mathcal{R}$ be the ring $\mathbb{Z}_p/F(X)$ where $F(X)$ is a degree-$n$ polynomial that splits entirely, and $p > n$.

**Ring-LPN assumption:** $(a, b) \sim (a, a \cdot e + f)$ where $(a, b) \leftarrow \mathcal{R}$ and $(e, f)$ are random $t$-sparse polynomials.

**Observation:** we can get $n$ OLE correlations from a single ‘ring-OLE’ correlation $(x, y, \langle x \cdot y \rangle)$ over $\mathcal{R}$: the OLE correlations are obtained by reducing $x$, $y$, and $x \cdot y$ modulo each of the linear factors $F_i$ of $F$.

**Construction:**
- Alice gets a pseudorandom polynomial $x = a \cdot e_x + f_x$ where $(e_x, f_x)$ are $t$-sparse polynomials over $\mathcal{R}$
- Bob gets a pseudorandom vector $y = a \cdot e_y + f_y$ where $(e_y, f_y)$ are $t$-sparse polynomials over $\mathcal{R}$
- Alice and Bob get shares of $x \cdot y = a^2 \cdot (e_x e_y) + a \cdot (e_x f_y + f_x e_y) + f_x f_y$

The polynomials $a^2$, $a$ are public, and $e_x e_y, e_x f_y, f_x e_y, f_x f_y$ are all $t^2$-sparse polynomials

Costs only $O(n \cdot \log n)$

- ‘Splittable ring-LPN’ deserves further study
- $\mathbb{F}$ must be large!
OLE Correlations, from Quasi-Abelian Syndrome Decoding

How do we break this ‘field-size barrier’? An answer in our recent Crypto paper (Bombar-C-Couvreur-Ducros’23): we move to quasi-abelian codes, which are defined over group algebras.

**High level intuition**

The group algebra structure gives a suitable framework to find the right polynomial $P$ to instantiate an LPN variant over a ring $\mathcal{R} = \mathbb{F}[X_1, \ldots, X_d]/P(X_1, \ldots, X_d)$ such that

- $\mathcal{R} \sim \mathbb{F} \times \cdots \times \mathbb{F}$ (i.e. we get many copies of an OLE over $\mathbb{F}$)
- The underlying assumption is plausibly secure (i.e. resists linear attacks)

Using multivariate rings gives us many more roots of $P$ even for a small $\mathbb{F}$! In fact, we can get up to $(|\mathbb{F}| - 1)^d$ copies of an OLE over $\mathbb{F}$.
OLE Correlations, from Quasi-Abelian Syndrome Decoding

How do we break this ‘field-size barrier’? An answer in our recent Crypto paper (Bombar-C-Couvreur-Ducros’23): we move to quasi-abelian codes, which are defined over group algebras.

High level intuition

The group algebra structure gives a suitable framework to find the right polynomial $P$ to instantiate an LPN variant over a ring $\mathcal{R} = \mathbb{F}[X_1, \ldots, X_d]/P(X_1, \ldots, X_d)$ such that

- $\mathcal{R} \sim \mathbb{F} \times \cdots \times \mathbb{F}$ (i.e. we get many copies of an OLE over $\mathbb{F}$)
- The underlying assumption is plausibly secure (i.e. resists linear attacks)

Using multivariate rings gives us many more roots of $P$ even for a small $\mathbb{F}$! In fact, we can get up to $(|\mathbb{F}| - 1)^d$ copies of an OLE over $\mathbb{F}$.

This only gives something meaningful up to $\mathbb{F}_3$!
Some of my Favourite Open Questions

- Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?
- Can we go below $s/\log \log s$?
- Can we build 3-party PCGs?
- How fast can PCGs be?

The barrier
The $\mathbb{F}_2$ barrier
The 2-party barrier
The quest for the perfect code
Public key PCFs: MPC as easy as key exchange
The communication of MPC with correlated randomness
Some of my favourite open question
Some of my Favourite Open Questions

- Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?
- Can we build 3-party PCGs?
- Can we go below $s/\log \log s$?
- How fast can PCGs be?
- Can we get fast, usable, scalable MPC over the internet?
A Closer Look at Secure Communication

Our ultimate goal is *practical* MPC that can be deployed and used over the web

Secure communication is already widely deployed and in use

> 85% of the total internet traffic is encrypted

Let us look at secure communication’s recipe for success!
A Closer Look at Secure Communication

Two Phases:

Key exchange phase

- One-time, *simultaneous* interaction
- Heavy (public key) computations
- Low communication $n \cdot |\mathcal{A}|$ (not $n^2$)

Encryption phase

- Lightweight (symmetric) computations
- Optimal message-to-cipher ratio
A Closer Look at Secure Communication

Two Phases:

Key exchange phase
- One-time, *simultaneous* interaction
- Heavy (public key) computations
- Low communication $n \cdot |\mathcal{K}|$ (not $n^2$)

Encryption phase
- Lightweight (symmetric) computations
- Optimal message-to-cipher ratio
Using a PRG enables a one-time generation of a fixed amount of correlations.
A pseudorandom correlation function is to a PCG what a PRF is to a PRG.

\[ \text{Function secret sharing} \quad \text{Short seed} \]

**Public function**

\[ FSS(C \circ \text{PRF}(\text{seed})) \]
A pseudorandom correlation function is to a PCG what a PRF is to a PRG.

Public function

\[ \text{FSS}(C \circ \text{PRF}(\text{seed})) \]

Function secret sharing \hspace{1cm} Short seed

A pseudorandom correlation *function* is to a PCG what a PRF is to a PRG.

Are there any FSS-friendly PRFs?
A pseudorandom correlation function is to a PCG what a PRF is to a PRG.

Public function

\[ FSS(C \circ PRF(\text{seed})) \]

Function secret sharing  Short seed

A pseudorandom correlation function is to a PCG what a PRF is to a PRG.

Are there any FSS-friendly PRFs?

The existence of PRFs in low complexity classes yields strong limitations for learning theory.

FOCS:BCGIKS20 and Crypto:BCGIKRS22 give plausible candidates.
Correctness & security:

- Black-box access to samples of the form $(F_{K_A}(x), F_{K_B}(x))$ are indistinguishable from black-box access to random samples from a target correlation.
- From the viewpoint of Alice, each $F_{K_B}(x)$ is indistinguishable from a random value sampled *conditioned on satisfying the correlation with $F_{K_A}(x)$.*
- Same condition in the other direction.
Public-Key Pseudorandom Correlation Functions

**Correctness & security:**

- Black-box access to samples of the form \((F_{KA}(x), F_{KB}(x))\) are indistinguishable from black-box access to random samples from a target correlation.
- From the viewpoint of Alice, each \(F_{KB}(x)\) is indistinguishable from a random value sampled conditioned on satisfying the correlation with \(F_{KA}(x)\).
- Same condition in the other direction.

**Formally:**

- \(\text{KeyGen} \rightarrow (pk, sk)\) generates public and private PCF keys
- \(\text{KeyDer}(pk_A, sk_B) \rightarrow K_{AB}^B\) yields Bob’s PCF key w.r.t. Alice’s key
- \(\text{Eval}(K, x) \rightarrow y\) yields a pseudorandom sample
Public-key PCFs are exactly the right tool to enable scalable, on-demand 2-party secure computation over the Internet, with a communication and computation pattern close to that of secure communication over the web.

Building efficient public-key PCF is essentially a wide-open question: the recent work of EC:Orlandi-Scholl-Yakoubov’21 gets it for OT from QR, but efficiency is quite bad.

(Teaser) Coming soon: we have some exciting progress in this line of work, which does not fully solve the problem, but is a big step forward!
Some of my Favourite Open Questions

- Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?
- Can we go below $s/\log \log s$?
- Can we get fast, usable, scalable MPC over the internet?
- Can we build 3-party PCGs?
- How fast can PCGs be?
- The communication of MPC with correlated randomness
- The $\mathbb{F}_2$ barrier
- The quest for the perfect code
- Public key PCFs: MPC as easy as key exchange
- Some of my favourite open question
Some of my Favourite Open Questions

- Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?
- Can we build 3-party PCGs?
- How fast can PCGs be?
- Can we get fast, usable, scalable MPC over the internet?
- Can we go below $s/\log \log s$?

The communication of MPC with correlated randomness
Some of my Favourite Open Questions

No time left for that, but I’d be happy to discuss it over dinner tonight!

Other cool things to check out that I don’t have time to discuss:

• People have been doing great things in zero-knowledge using these PCG techniques (incl. right here in Aarhus!)
• Everything we have so far works only for two parties!
• … And many more
Questions?