Efficient Pseudorandom Correlation Generators: Silent OT Extension and More (BCG+19b)

想法概述：

在MPC中经常用到correlated randomness，譬如OT，但是有效率瓶颈，尤其是communication和storage。
那么想法就是把一部分转化为local计算，这样online计算时communication就会减少。

> PCG的直观想法和安全性：
> A dream goal would be to replace this source of correlated randomness with short correlated seeds, which can be “silently” expanded without any interaction to produce a large amount of pseudorandom correlated randomness. This process should emulate an ideal process for generating the target correlation not only from the point of view of outsiders, but also from the point of view of insiders who can observe the correlated seeds. We refer to such an object as a pseudorandom correlation generator, or PCG for short

有很多种在MPC中有用的correlated randomness：

- oblivious linear-function evaluation (OLE) correlations
- vector OLE (VOLE) correlation
- subfield VOLE （sVOLE）
- multiplication triples (also known as “Beaver triples”)
- one-time truth tables （OTTT）
- 在（BCG+20）中还会讨论很多authenticated版本的correlation
- 一些generalized概念：bilinear correlation、degree-d correlation

之前的工作的构造绝大部分都是从实用的角度讲不高效的，一个还可以的例子就是（BCGI18）也是本文最主要的参考文献。

> In particular, there was no prior approach (even a heuristic one) for constructing a concretely efficient PCG for OT correlations.

本文工作概述：

- 给出一个PCG的定义，并且排除掉最直观的最显然的一种simulation-based定义，因为这种定义是无法达成的。
  > Unfortunately, we show that such a definition is impossible to realize even for simple correlations. Intuitively, the impossibility follows from the fact that in the real distribution k_b

> "explains" the output of the honest party in an efficiently verifiable way, whereas such an explanation of r_{1-b} cannot be generated from r_b in the ideal distribution.

- 本文给出的PCG定义也不是万能的，但是提出一种MPC protocol with stronger security requirement，可以plug-and-play的使用我们定义的PCG。概括来说，是允许adversary自己选取自己的randomness然后发送给honest party他根据correlation选取的randomness。
  > It fortunately turns out that natural MPC protocols in the preprocessing model already satisfy this stronger security requirement.

- PCG和HSS相互蕴含
  > In particular, HSS for general circuits implies PCG for all additive correlations, which include most of the useful MPC correlations as special cases. (This is only a feasibility result, which does not directly imply concretely efficient constructions.)

- 本文最主要结果之一："the first concretely efficient construction of PCG for the oblivious transfer (OT) correlation"。除了用了LPN-assumption，还用了correlation-robust hash function in a black-box way, 这个hash实际中可以用general-purpose hash function or block cipher实现)
  > we obtain a silent OT extension protocol that generates n pseudo-random OT instances using a small number of OTs, with a total of $O(n^{\varepsilon})$ bits of communication for any $\varepsilon > 0$. This should be compared with existing OT extension protocols [Bea96,IKNP03] that do not require the LPN assumption but where the communication complexity is bigger than n.

- 很高效：Concrete Efficiency. Our LPN-based PCG for OT is very attractive in terms of concrete efficiency, and we expect it to outperform state-of-the-art OT extension protocols [IKNP03, ALSZ13,KK13] in settings where communication is the bottleneck.
- 一些其他的2-party correlation 构造（基于各种各样的assumption）：
  - PCG for Constant-Degree Polynomials from LPN.
    > The main caveat is that even for generating simple degree-d correlations, such as $\Omega(n)$ Beaver triples $(d = 2)$, the computational complexity of Expand is bigger than n^d, much slower than our PCG for OT

  - PCG for One-Time Truth Tables from any PRG.
    > We present a very simple PCG for authenticated OTTT using only a distributed point function (DPF) [GI14,BGI16b], which in turn can be efficiently constructed from any pseudoran- dom generator (PRG). This PCG follows naturally from a building block of the silent OT extension construction. It compresses the storage cost of an authenticated OTTT from $O(\lambda n)$ bits down to $O(\lambda \log n)$ bits, for a table of size n

  - PCGs from Homomorphic Secret Sharing:
    给出 practically feasible PCG for OLE and (authenticated) Beaver triple correlations(which

are useful for arithmetic MPC protocols such as SPDZ [DPSZ12]). 在这些构造中所用的HSS基于 ring-LWE [BGV12, DHRW16, BKS19] and the BGN (pairing-based) cryptosystem [BGN05,BGI16a,BCG+17]. To expand the seeds, we rely on a multivariate quadratic (MQ) assumption based PRG, which limits the stretch to sub-quadratic, but allows for reasonable computational efficiency. 虽然这种相对于silence-OT 很慢，但是有一个优势：可以构造multi-party correlation.

- PCG for multi-party correlation:

  Finally, we present a general transformation for extending certain classes of PCGs from the 2-party to the multi-party setting. This can be applied to PCGs for simple bilinear correlations, including VOLE and Beaver triples, giving the first non-trivial, efficient PCG constructions in the multi-party setting. The transformation applies to most of our 2-party PCGs, including the LPN-based PCG for constant-degree correlations.

  On top of the silent preprocessing feature, an appealing application of our multi-party PCGs is in obtaining secure M-party computation protocols with *total* communication complexity $O(Ms + M^2 \cdot s^\epsilon)$ (for circuit size s and constant $0 < \epsilon < 1$). The $O(Ms)$ term is the cost of the (information-theoretic) online phase, and the $O(M^2 \cdot s^\epsilon)$ term is the cost of distributing the PCG seed generation, which is the only part of the protocol requiring pairwise communication. This should be contrasted with OT-based MPC protocols, which have total communication complexity $\Omega(M^2 s)$ [GMW87,HOSS18] . Protocols with such communication complexity (without the silent preprocessing feature) could previously be based on different flavors of somewhat homomorphic encryption [FH96, CDN01, DPSZ12]. We get the first such protocol that only relies on LPN and OT, and the first practically feasible protocol that has sublinear-communication offline phase and information-theoretic online phase.

各种PCG构造的总结表格：

PCG	Section 5	Section B	Section 6	Section 4.4	Sections 7.3,C,D,E	Sections 7.4,F
Assumption	LPN	PRG*	LPN	deg-d HSS + MQ/LPN	SXDH + LPN	LWE + MQ
Correlations	OT*	OTTT*	deg-d	degree-$d/2$	small-ring deg-2	deg-d
Efficiency	1M OT/s†	-	-	-	5 OLE/s‡	7000 ABT/s**
Multiparty (bilinear corr.)	✗	✗	✔	✔	✗	✔

各个section之间关系结构：

1. section2: technical overview
2. section3: Preliminary
3. section4: PCG definition and foundational results
4. section5: 构造silence OT extension 以及一些其他应用（如NIZK）
5. section6: LPN-based construction of PCG for general constant-degree correlations
6. 从section4.4的基于PRG构造PCG的一般方法，
   到section7.3和7.4中基于group/lattice的HSS构造更复杂correlation。

7. section8: 基于programmable 2-party PCG构造multi-party PCGs for simple bilinear correlations

Preliminar：

Notation:

mial. For two families of distributions $X = \{X_\lambda\}$ and $Y = \{Y_\lambda\}$ indexed by a security parameter $\lambda \in \mathbb{N}$, we write $X \overset{c}{\approx} Y$ if X and Y are *computationally indistinguishable* (namely, any family of circuits of size $\mathsf{poly}(\lambda)$ has a negligible distinguishing advantage), $X \overset{s}{\approx} Y$ if they are *statistically indistinguishable* (namely, the above holds for arbitrary distinguishers), and $X \equiv Y$ if the two families are identically distributed.

Funciton Secret Sharing (FSS):

Definition 1 (Function Secret Sharing; adapted from [BGI16b]). *A 2-party* function secret sharing *(FSS) scheme for a class of functions $\mathcal{C} = \{f : I \to \mathbb{G}\}$ with input domain I and output domain an abelian group $(\mathbb{G}, +)$, is a pair of PPT algorithms $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ with the following syntax:*

- $\mathsf{FSS.Gen}(1^\lambda, f)$*, given security parameter λ and description of a function $f \in \mathcal{C}$, outputs a pair of keys (K_0, K_1);*
- $\mathsf{FSS.Eval}(b, K_b, x)$*, given party index $b \in \{0, 1\}$, key K_b, and input $x \in I$, outputs a group element $y_b \in \mathbb{G}$.*

Given an allowable leakage function $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$, the scheme FSS should satisfy the following requirements:

- **Correctness:** *For any $f : I \to \mathbb{G}$ in $\mathcal{C}$ and $x \in I$, we have $\Pr[(K_0, K_1) \overset{\$}{\leftarrow} \mathsf{FSS.Gen}(1^\lambda, f) : \sum_{b\in\{0,1\}} \mathsf{FSS.Eval}(b, K_b, x) = f(x)] = 1$.*
- **Security:** *For any $b \in \{0, 1\}$, there exists a PPT simulator Sim such that for any polynomial-size function sequence $f_\lambda \in \mathcal{C}$, the distributions $\{(K_0, K_1) \overset{\$}{\leftarrow} \mathsf{FSS.Gen}(1^\lambda, f_\lambda) : K_b\}$ and $\{K_b \overset{\$}{\leftarrow} \mathsf{Sim}(1^\lambda, \mathsf{Leak}(f_\lambda))\}$ are computationally indistinguishable.*

Unless otherwise specified, we assume that for $f : I \to \mathbb{G}$, the allowable leakage $\mathsf{Leak}(f)$ outputs $(I, \mathbb{G})$, namely a description of the input and output domains of f.

Some applications of FSS require applying the evaluation algorithm on *all inputs*. Following [BGI16b, BCGI18], given an FSS scheme $(\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$, we denote by $\mathsf{FSS.FullEval}$ an algorithm which, on input a bit b, and an evaluation key K_b (which defines the input domain I), outputs a list of $|I|$ elements of $\mathbb{G}$ corresponding to the evaluation of $\mathsf{FSS.Eval}(b, K_b, \cdot)$ on every input $x \in I$ (in some predetermined order). While $\mathsf{FSS.FullEval}$ can always be realized with $|I|$ invocations of $\mathsf{FSS.Eval}$, it is typically possible to obtain a more efficient construction. Below,

安全性的直观：除了f的输入和输出大小，key computationly 隐藏f。

一个重要性质：某些情况下evaluation的输出有伪随机性：

Remark 2. In any FSS scheme for a sufficiently rich class of functions (including point functions), each of the two evaluation functions $F_K^b(x) = \mathsf{FSS.Eval}(b, K, x)$ is a pseudorandom function [BGI15]. Some of our constructions will use this property.

两个特殊的FSS：

1. distributed point functions（DPF）：可以由PRG构造：

   A distributed point function (DPF) [GI14] is an FSS scheme for the class of point functions $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathbb{G}$ which satisfy $f_{\alpha,\beta}(\alpha) = \beta$, and $f_{\alpha,\beta}(x) = 0$ for any $x \neq \alpha$. A sequence of works [GI14, BGI15, BGI16b] has led to highly efficient constructions of DPF schemes from any pseudorandom generator (PRG), which can be implemented in practice using block ciphers such as AES.

   Theorem 3 (PRG-based DPF [BGI16b], Theorems 3.3 and 3.4). *Given a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, there exists a DPF for point functions $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathbb{G}$ with key size $\ell \cdot (\lambda+2) + \lambda + \lceil \log_2 |\mathbb{G}| \rceil$ bits. For $m = \lceil \frac{\log |\mathbb{G}|}{\lambda+2} \rceil$, the key generation algorithm* Gen *invokes G at most $2(\ell+m)$ times, the evaluation algorithm* Eval *invokes G at most $\ell+m$ times, and the full evaluation algorithm* FullEval *invokes G at most $2^\ell(1+m)$ times.*

2. multi-point function secret sharing（MPFSS）：

   Definition 4 (Multi-Point Function [BCGI18]). *An (n,t)-multi-point function over an abelian group $(\mathbb{G},+)$ is a function $f_{S,\boldsymbol{y}} : [n] \to \mathbb{G}$, where $S = (s_1, \cdots, s_t)$ is an ordered subset of $[n]$ of size t and $\boldsymbol{y} = (y_1, \cdots, y_t) \in \mathbb{G}^t$, defined by $f_{S,\boldsymbol{y}}(s_i) = y_i$ for any $i \in [t]$, and $f_{S,y}(x) = 0$ for any $x \in [n] \setminus S$.*

   We assume that the description of S includes the input domain $[n]$ so that $f_{S,\boldsymbol{y}}$ is fully specified.

   可以直接把t个DPF加在一起构造，也可以用“batch code”加速，见（BCGI8）。

Homomorphic Secret Sharing（HSS）：

理解为FSS的对偶版本，或者secret sharing版本的FHE。

我们一般只关心这样的HSS： low-degree multivariate polynomials on shared input vectors

安全性直观：有evaluation key和share可以隐藏输入x。

环$\mathcal{R}$是内蕴在security parameter λ 中的，元素的bit-length至多多项式于λ。

Definition 5 (Degree-d Homomorphic Secret Sharing). *A (2-party, secret-key)* Degree-d Homomorphic Secret Sharing (HSS) *scheme over a ring* $(\mathcal{R} = \mathcal{R}(\lambda), +, \cdot)$ *is a triple of PPT algorithms* $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *with the following syntax:*

- $\mathsf{HSS.Gen}(1^\lambda)$*: On input a security parameter* 1^λ*, the key generation algorithm outputs a secret key* sk *and an evaluation key* ek*.*
- $\mathsf{HSS.Share}(\mathsf{sk}, x)$*: Given secret key* sk *and secret input value* $x \in \mathcal{R}^n$*, the sharing algorithm outputs a pair of shares* (s_0, s_1)*. We assume that a description of the ring* $\mathcal{R}$ *and the input length* n *are included in each of* (s_0, s_1)*.*
- $\mathsf{HSS.Eval}(b, \mathsf{ek}, s_b, P)$*: On input party index* $b \in \{0, 1\}$*, evaluation key* ek*, share* s_b *of an input vector* $x \in \mathcal{R}^n$*, and degree-*d *arithmetic circuit* P *over* $\mathcal{R}$ *with* n *inputs and* m *outputs, the (deterministic) homomorphic evaluation algorithm outputs* $y_b \in \mathcal{R}^m$*, constituting party* b*'s share over* $\mathcal{R}$ *of an output* $y \in \mathcal{R}^m$*.*

The algorithms $(\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *should satisfy the following correctness and security requirements:*

- **Correctness:** *For every polynomial* $\mathsf{poly}(\lambda)$ *there exists a negligible* $\mathsf{negl}(\lambda)$ *such that for every* λ*, input* $x \in \mathcal{R}^n$ *(where* $\mathcal{R} = \mathcal{R}(\lambda)$*), and degree-*$d$ *arithmetic circuit* P *of size* $\mathsf{poly}(\lambda)$ *we have:*

$$\Pr[y_0 + y_1 \neq P(x)] \leq \mathsf{negl}(\lambda),$$

  where probability is taken over

$$(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{HSS.Gen}(1^\lambda); (s_0, s_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, x);$$
$$y_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{ek}, s_b, P),\ b \in \{0, 1\}.$$

- **Security:** *For any* $b \in \{0, 1\}$*, pair of input sequences* $x_\lambda, x'_\lambda \in \mathcal{R}^n$ *of polynomial length* $n(\lambda)$*, the distribution ensembles* $C_b(\lambda, x_\lambda)$ *and* $C_b(\lambda, x'_\lambda)$ *are computationally indistinguishable, where* $C_b(\lambda, z)$ *for* $z \in \{x_\lambda, x'_\lambda\}$ *is obtained by sampling* $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{HSS.Gen}(1^\lambda)$*, sampling* $(s_0, s_1) \leftarrow \mathsf{HSS.Enc}(\mathsf{sk}, z)$*, and outputting* (ek, s_b)*.*

Learning Parity with Noise（LPN）：

Definition 6 (LPN). *Let* $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_{k,q}(\mathcal{R})\}_{k,q \in \mathbb{N}}$ *denote a family of distributions over a ring* $\mathcal{R}$*, such that for any* $k, q \in \mathbb{N}$*,* $\mathsf{Im}(\mathcal{D}_{k,q}(\mathcal{R})) \subseteq \mathcal{R}^q$*. Let* $\mathbf{C}$ *be a probabilistic code generation algorithm such that* $\mathbf{C}(k, q, \mathcal{R})$ *outputs a matrix* $A \in \mathcal{R}^{k \times q}$*. For dimension* $k = k(\lambda)$*, number of samples (or block length)* $q = q(\lambda)$*, and ring* $\mathcal{R} = \mathcal{R}(\lambda)$*, the* $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-$\mathsf{LPN}(k, q)$ *assumption states that*

$$\{(A, \boldsymbol{b}) \mid A \xleftarrow{\$} \mathbf{C}(k, q, \mathcal{R}), \boldsymbol{e} \xleftarrow{\$} \mathcal{D}_{k,q}(\mathcal{R}), \boldsymbol{s} \xleftarrow{\$} \mathbb{F}^k, \boldsymbol{b} \leftarrow \boldsymbol{s} \cdot A + \boldsymbol{e}\}$$
$$\overset{c}{\approx} \{(A, \boldsymbol{b}) \mid A \xleftarrow{\$} \mathbf{C}(k, q, \mathcal{R}), \boldsymbol{b} \xleftarrow{\$} \mathcal{R}^q\}$$

参数都是λ的函数，并且computational indistinguishable也是基于λ定义。

接下来我们集中考虑error为固定hamming weight的分布情况的LPN：

For a finite field $\mathbb{F}$, we denote by $\mathcal{HW}_r(\mathbb{F})$ the distribution of uniform, weight r vectors over $\mathbb{F}$; that is, a sample from $\mathcal{HW}_r(\mathbb{F})$ is a uniformly random nonzero field element in r random positions, and zero elsewhere. The $(\mathsf{Ber}_r(\mathbb{F})^q, \mathbf{C}, \mathbb{F}) - \mathsf{LPN}(k, q)$ assumption corresponds to the standard (non-binary, fixed-weight) LPN assumption over a field $\mathbb{F}$ with code generator $\mathbf{C}$, dimension k, number of samples (or block length) q, and noise rate r.

注意到，如果q和r的选择使得k个随机变量大概率都没有error，那么就可以用高斯消元法求解，所以一般要求$q = O(k)$。

Definition 7 (dual LPN). *Let $\mathcal{D}(\mathcal{R})$ and $\mathbf{C}$ be as in Definition 6, $n, n' \in \mathbb{N}$ with $n' > n$, and define $\mathbf{C}^{\perp}(n', n, \mathcal{R}) = \{B \in \mathcal{R}^{n' \times n} : A \cdot B = 0, A \in \mathbf{C}(n' - n, n', \mathcal{R}), \mathsf{rank}(B) = n\}$.*

For $n = n(\lambda), n' = n'(\lambda)$ and $\mathcal{R} = \mathcal{R}(\lambda)$, the $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-dual-LPN$(n', n)$ assumption states that

$$\{(H, \boldsymbol{b}) \mid H \xleftarrow{\$} \mathbf{C}^{\perp}(n', n, \mathcal{R}), \boldsymbol{e} \xleftarrow{\$} \mathcal{D}(\mathcal{R}), \boldsymbol{b} \leftarrow \boldsymbol{e} \cdot H\}$$
$$\overset{c}{\approx} \{(H, \boldsymbol{b}) \mid H \xleftarrow{\$} \mathbf{C}^{\perp}(n', n, \mathcal{R}), \boldsymbol{b} \xleftarrow{\$} \mathcal{R}^n\}$$

The search version of the dual LPN problem is also known as syndrome decoding. The decision version defined above is equivalent to primal variant of LPN from Definition 6 with dimension $k = n' - n$ and number of samples $q = n'$. This follows from the simple fact that $(\boldsymbol{s} \cdot A + \boldsymbol{e}) \cdot H = \boldsymbol{s} \cdot A \cdot H + \boldsymbol{e} \cdot H = \boldsymbol{e} \cdot H$, when H is the parity-check matrix of A.

注意，要保证安全，必须overwhelming probability H满秩，否则输出有线性关系。

regular-LPN or regular syndrome decoding problem or regular error distribution: 把整个error向量平分成若干块，每块有且仅有一个随机的位置有error。

Pseudorandom Correlation Generators 定义：

结构：

1. section4.1: PCG的正式定义
2. section4.2: 说明simulation-based定义虽然可以用在任何的MPC中，但是不能实现。
3. section4.3: 我们的定义可以在某类加强安全性的MPC中使用，事实上被自然的MPC所满足
4. section4.4: two-way relation between PCGs for a useful class of “low-degree correlations” and HSS for low-degree polynomials

定义：

Definition 10 (Correlation Generator). *A PPT algorithm $\mathcal{C}$ is called a* correlation generator, *if $\mathcal{C}$ on input 1^λ outputs a pair of elements in $\{0,1\}^n \times \{0,1\}^n$ for $n \in \mathsf{poly}(\lambda)$.*

Definition 11 (Reverse-sampleable Correlation Generator). *Let $\mathcal{C}$ be a correlation generator. We say $\mathcal{C}$ is* reverse sampleable *if there exists a PPT algorithm* RSample *such that for $\sigma \in \{0,1\}$ the correlation obtained via:*

$$\{(R'_0, R'_1) \,|(R_0, R_1) \xleftarrow{\$} \mathcal{C}(1^\lambda), R'_\sigma := R_\sigma, R'_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)\}$$

is computationally indistinguishable from $\mathcal{C}(1^\lambda)$.

Definition 12 (Pseudorandom Correlation Generator (PCG)). *Let $\mathcal{C}$ be a reverse-sampleable correlation generator. A* pseudorandom correlation generator (PCG) for $\mathcal{C}$ *is a pair of algorithms* $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *with the following syntax:*

- $\mathsf{PCG.Gen}(1^\lambda)$ *is a PPT algorithm that given a security parameter λ, outputs a pair of seeds $(\mathsf{k}_0, \mathsf{k}_1)$;*
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ *is a polynomial-time algorithm that given party index $\sigma \in \{0,1\}$ and a seed k_σ, outputs a bit string $R_\sigma \in \{0,1\}^n$.*

The algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *should satisfy the following:*

- **Correctness.** *The correlation obtained via:*

$$\{(R_0, R_1) \,|(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma) \text{ for } \sigma \in \{0,1\}\}$$

  is computationally indistinguishable from $\mathcal{C}(1^\lambda)$.
- **Security.** *For any $\sigma \in \{0,1\}$, the following two distributions are computationally indistinguishable:*

$$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \text{ and}$$
$$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_{1-\sigma}), R_\sigma \xleftarrow{\$} \mathsf{RSample}(\sigma, R_{1-\sigma})\}$$

  where RSample *is the reverse sampling algorithm for correlation $\mathcal{C}$.*

安全性的直观理解：adversary得到自己手中的randomness后无助于得知另一个人的randomness，除了知道是correlated的。

带有setup的PCG：一个准备阶段，可以在多次PCG instance中使用：

Remark 13 (PCG with Setup). We sometimes consider an additional algorithm $\mathsf{PCG.Setup}$ to sample a secret key, public parameters and a share of evaluation keys (or a subset of the mentioned), which can be reused throughout several instances. More precisely: On input 1^λ, $\mathsf{PCG.Setup}$ returns a tuple $(\mathsf{pp}, \mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}})$, $\mathsf{PCG.Gen}$ receives the secret key sk as additional input (always assumed to include the public parameters pp), and $\mathsf{PCG.Expand}$ receives the public parameters pp and the respective evaluation key share ek_σ as additional inputs.

Multi-Party Setting：定义完全相同：

Remark 14 (PCG in the Multi-Party Setting). We also consider multi-party PCGs for reverse sampleable multi-correlation generators $\mathcal{C}^M$ which on input 1^λ outputs elements in $(\{0,1\}^n)^M$. In this case, $\mathsf{PCG.Gen}(1^\lambda)$ returns a M-tuple $(\mathsf{k}_1, \ldots, \mathsf{k}_M)$. Correctness is defined accordingly and security required against any subset of colluding parties. More precisely: For any $T \subset \{1, \ldots, M\}$, we require the following two distributions to be computationally indistinguishable:

$$\begin{aligned}
&\{(\{\mathsf{k}_j\}_{j\in T}, \{R_i\}_{i\notin T}) \mid (\mathsf{k}_1, \ldots, \mathsf{k}_M) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda),\\
&\qquad\qquad \forall i \notin T\colon R_i \leftarrow \mathsf{PCG.Expand}(i, \mathsf{k}_i)\} \quad \text{and}\\
&\{(\{\mathsf{k}_j\}_{j\in T}, \{R_i\}_{i\notin T}) \mid (\mathsf{k}_1, \ldots, \mathsf{k}_M) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda),\\
&\qquad\qquad \forall j \in T\colon R_j \leftarrow \mathsf{PCG.Expand}(j, \mathsf{k}_j),\\
&\qquad\qquad \{R_i\}_{i\notin T} \xleftarrow{\$} \mathsf{RSample}(T, \{R_j\}_{j\in T})\}
\end{aligned}$$

Simulation-Based 定义的不可行性：

直观想法：理想是每个MPC protocol中的correlated randomness我们都可以直接用seed生成的CR代替。但是我们考虑一个最简单的protocol，第一个人生成CR (R_1, R_2)，然后直接把R_2给第二个人，如果我们直接换成第一个人generate两个seed (k_1, k_2)，那么simulation就相当于知道R_1后找到k_1，这样是有悖于正确性的：

> Simulating the above protocol given only the output R1 corresponds to finding a short seed k1 that can be (deterministically) expanded to R1. If the entropy in the second output of C exceeds the seed-length |k1|, such a compression violates correctness, as it could be used to distinguish R1 from a string that is indeed chosen via C.

引入必要的新概念，Yao Incompressiblity Entropy：

直观理解：对分布的输出可以压缩到什么程度的度量：

More precisely, Yao incompressibility entropy [HLR07,Yao82] is a measure on how well outputs of a distribution can be compressed on average, when the compressing and decompressing algorithms are required to be efficient. For example, a pseudorandom bit string of length l has Yao incompressibility entropy l.

Definition 15 (Yao Incompressiblity Entropy [HLR07] (simplified)). *Let $\ell = \ell(\lambda) \in \mathbb{N}$. A probability ensemble $X = \{X_\lambda\}$ has* Yao incompressibility entropy at least ℓ*, if for every pair of polynomial sized circuit-ensembles $C = \{C_\lambda\}$, $D = \{D_\lambda\}$ where C has output bit-length at most $\ell - 1$, there exists a negligible function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}^+$ such that for every sufficiently large positive integer λ we have*

$$\Pr\left[x \leftarrow X\colon D(C(x)) = x\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Theorem 16 (Impossiblity of Simulation-Based Definition for Non-Trivial PCGs). *Let $\mathcal{C}$ be a reverse-sampleable correlation generator, where the Yao incompressibility entropy of the output is ℓ. Then, for every pseudorandom correlation generator $\mathsf{PCG} = (\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ satisfying simulation-based security, the output of the seed generation $\mathsf{PCG.Gen}$ algorithm must at least have bit-length ℓ.*

Proof. Let $\mathsf{PCG} = (\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ be a pseudorandom correlation generator for $\mathcal{C}$ that satisfies simulation-based security. Then, in particular, the following protocol Π_{PCG} has to satisfy one-sided security against P_1: Party P_0 runs $(k_0, k_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda)$ and sends k_1 to P_1. Finally, P_1 outputs $R_1 \leftarrow \mathsf{PCG.Expand}(1, k_1)$.

Let ℓ_1 be the Yao incompressibility entropy of the output of $\mathcal{C}^1(1^\lambda) := \{R_1 \mid (R_0, R_1) \leftarrow \mathcal{C}(1^\lambda)\}$. Further, let

$$\mathcal{C}^1_{\mathsf{PCG}}(1^\lambda) := \{R_1 \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_1 \leftarrow \mathsf{PCG.Expand}(1, \mathsf{k}_1)\}.$$

By correctness of the PCG, the output of $\mathcal{C}^1_{\mathsf{PCG}}$ (and therefore the output of the protocol Π_{PCG}) must meet the Yao incompressibility entropy ℓ_1, as an efficient pair of compressor and decompressor could be used as a distinguisher between $\mathcal{C}^1$ and $\mathcal{C}^1_{\mathsf{PCG}}$.

By [HW15, Theorem 5], for any protocol between two parties P_0 and P_1 with one-sided security against "honest-but-deterministic"[11] P_1, where P_1 has no input, it holds: *If the Yao incompressibility entropy of the output of P_1 is ℓ_1, then the communication complexity from P_0 to P_1 must be at least ℓ_1 bits.*

Therefore, as seed expansion is deterministic, the bit-length $|k_1|$ of the seed of the second party must be at least ℓ_1. Reversing the roles of P_0 and P_1 together with additivity of Yao incompressibility entropy yields the required.

新定义的使用：

我们的PCG可以直接使用在含有弱形式CR的protocol，其中corrupted人可以影响他们的输出

具体地说，我们要实现如下的functionality，其中corrupted party可以选择自己的output，然后honest party根据这个输出做reverse sampling：

Functionality $\mathcal{F}^{\mathcal{C}}_{\mathsf{corr}*}$

On input 1^λ, the functionality does as follows:

- If no parties are corrupt, sample $(R_0, R_1) \xleftarrow{\$} \mathcal{C}(1^\lambda)$.
- Otherwise, if P_σ is corrupt, wait to receive $R_\sigma \in \{0,1\}^{n_\sigma}$ from $\mathcal{A}$, then sample $R_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)$.

The functionality outputs R_0 to P_0 and R_1 to P_1, and then halts.

> 一个例子是random OT，问题：random OT和普通OT的关系？

To realize $\mathcal{F}^{\mathcal{C}}_{\mathsf{corr}*}$, we use a simple protocol, $\Pi^{\mathcal{C}}_{\mathsf{corr}*}$, that calls $\mathcal{F}^{\mathsf{PCG.Gen}}_{\mathsf{corr}}$ so that each party obtains a seed k_σ, which is then expanded to get the output $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$.

Theorem 19. *Let* $\mathsf{PCG} = (\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *be a secure PCG for a reverse-sampleable correlation generator,* $\mathcal{C}$*. Then the protocol* $\Pi_{\mathsf{corr}*}$ *securely realizes the* $\mathcal{F}^{\mathcal{C}}_{\mathsf{corr}*}$ *functionality against a static, malicious adversary.*

Proof. Let $\mathcal{A}$ be a static adversary against the protocol π. We construct a simulator Sim, which interacts with $\mathcal{A}$ and $\mathcal{F}^{\mathcal{C}}_{\mathsf{corr}*}$ to produce a view for $\mathcal{A}$ that is indistinguishable from a real execution of the protocol. When both parties are corrupted, the simulator just runs $\mathcal{A}$ internally and security is straightforward. Similarly, when both parties are honest, simulation is trivial and indistinguishability follows from the correctness of PCG. Now suppose that only P_σ is corrupted, for $\sigma \in \{0,1\}$. On receiving the input 1^λ, Sim samples a pair of seeds $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda)$, then sends k_σ to $\mathcal{A}$ as its output of $\mathcal{F}^{\mathsf{PCG.Gen}}_{\mathsf{corr}}$, computes $R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ and sends this to $\mathcal{F}^{\mathcal{C}}_{\mathsf{corr}*}$. Notice that in the ideal execution, the view of the distinguisher consists of the seed k_σ and the honest party's output $R_{1-\sigma}$, which is computed by $\mathcal{F}^{\mathcal{C}}_{\mathsf{corr}*}$ as $R_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)$. The only difference in the real execution, is that there the honest party's output is computed with $\mathsf{PCG.Expand}(1-\sigma, \mathsf{k}_{1-\sigma})$. These two views are computationally indistinguishable, due to the security property of PCG.

由HSS构造PCG：

想法是把PCG和HSS混合在一起。这里主要考虑additive correlation：

consider the special case of additive correlations, where R_0, R_1 are uniformly distributed subject to $R_0 + R_1 = f(X)$ for a random input X and fixed function f. Now, consider an HSS scheme

This gives rise to the following PCG construction: During key generation a short seed k is shared between the players (as HSS shares). For expansion, the players can then locally evaluate $f(\mathsf{PRG}(\mathsf{k}))$ via the HSS operations. By the correctness of the HSS that indeed gives outputs R_0, R_1 with $R_0 + R_1 = f(X)$, where $X = \mathsf{PRG}(\mathsf{k})$. In this section we formally prove that the described construction meets the PCG requirements.

主要的问题是效率，因为已知的比较高效的HSS都只针对于很有限的函数类，所以我们需要精心的选取PCG和HSS，这一问题将在section 7阐述。

下面是formalized定义：

首先是一种推广化的PRG：

Definition 20 ($\mathcal{D}^\ell$-Pseudorandom Generator). *Let $\mathcal{R}$ be a ring (parametrized implicitly by λ) and let $\mathcal{D}^\ell$ be a distribution on $\mathcal{R}^\ell$. We say* $\mathsf{PRG}\colon \mathcal{R}^\ell \to \mathcal{R}^n$ *is a* $\mathcal{D}^\ell$-pseudorandom generator (PRG)*, if the following two distributions are computationally indistinguishable:*

$$\left\{Y \mid X \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), Y := \mathsf{PRG}(X)\right\} \text{ and } \left\{Y \mid Y \xleftarrow{\$} \mathcal{U}^n(\mathcal{R})\right\}.$$

其次是一种additive correlation：

Definition 21 (Correlation Generators for Additive Correlations). *Let $\mathcal{R}$ be a ring. Let $n, m \in \mathbb{N}$ and $\mathcal{F} \subseteq \{f\colon \mathcal{R}^n \to \mathcal{R}^m\}$ be a family of functions. Then we define a* correlation generator $\mathcal{C}_\mathcal{F}$ for $\mathcal{F}$ *as follows: On input 1^λ and $f \in \mathcal{F}$ the correlation generator $\mathcal{C}_\mathcal{F}$ samples $X \xleftarrow{\$} \mathcal{U}^n(\mathcal{R})$, and returns a pair $(R_0, R_1) \in \mathcal{R}^m \times \mathcal{R}^m$, which is distributed uniformly at random conditioned on $R_0 + R_1 = f(X)$.*

再是HSS的一种性质，大意为每个party的输出为伪随机的：

Definition 22 (HSS satisfying Pseudorandomness of Outputs). *We say an HSS* $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *for a function family $\mathcal{F} := \{f\colon \mathcal{R}^n \to \mathcal{R}^m\}$ satisfies* pseudorandomness of outputs*, if for all $f\colon \mathcal{R}^n \to \mathcal{R}^m \in \mathcal{F}$, $(\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda)$, $X \xleftarrow{\$} \mathcal{U}^n(\mathcal{R})$, $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{Share}(\mathsf{sk}, X)$, and $\sigma \in \{0,1\}$ the output $R_\sigma \xleftarrow{\$} \mathsf{HSS.Eval}(\sigma, \mathsf{ek}_\sigma, \mathsf{k}_\sigma, f)$ is distributed computationally close to uniformly at random over the output space.*

Note that if $f(\mathcal{U}^n(\mathcal{R}))$ is close to being uniformly random on $\mathcal{R}^m$, this property follows from the security of HSS.

然后就是我们的构造：

- $\mathsf{PCG.Setup}(1^\lambda)$: Sample and output $(\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda)$.
- $\mathsf{PCG.Gen}(\mathsf{sk})$: Sample $r \leftarrow \mathcal{D}^\ell$ and output $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r)$.
- $\mathsf{PCG.Expand}(\sigma, \mathsf{ek}_\sigma, \mathsf{k}_\sigma, f)$: Output $R_\sigma \leftarrow \mathsf{HSS.Eval}(\sigma, \mathsf{ek}_\sigma, \mathsf{k}_\sigma, f \circ \mathsf{PRG})$.

Fig. 2. PCG for correlation $\mathcal{C}_\mathcal{F}$. Here, PRG is a $\mathcal{D}^\ell$-PRG and $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ an HSS for the family of functions $\mathcal{F}_{\mathsf{HSS}} := \{f \circ \mathsf{PRG}\colon r \mapsto f(\mathsf{PRG}(r)) \mid f \in \mathcal{F}\}$.

定理陈述和证明，一步步根据各种性质演变过去：

Theorem 23. (PCG for Additive Correlations from HSS). *Let $\mathcal{R}$ be a ring and $n, m, \ell \in \mathbb{N}$. Let $\mathcal{F} \subseteq \{f\colon \mathcal{R}^n \to \mathcal{R}^m\}$ be a family of functions. Let* PRG *be a $\mathcal{D}^\ell$-PRG and* $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *an HSS with overhead O_{HSS}*[12] *for the family of functions $\mathcal{F}_{\mathsf{HSS}} := \{f \circ \mathsf{PRG}\colon \mathcal{R}^\ell \to \mathcal{R}^m, r \mapsto f(\mathsf{PRG}(r)) \mid f \in \mathcal{F}\}$ that further satisfies pseudorandomness of outputs. Then,* $\mathsf{PCG} = (\mathsf{PCG.Setup}, \mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *as defined in Figure 2 is a* PCG *for the correlation generator $\mathcal{C}_{\mathcal{F}}$ with key-length upper bounded by $\ell \cdot O_{\mathsf{HSS}}$.*

Proof. Correctness. Let $f \in \mathcal{F}$. We have

$$
\begin{aligned}
&\{(R_0, R_1) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma) \text{ for } \sigma \in \{0,1\}\} \\
\overset{c}{\approx} &\{(R_0, R_1) \mid (\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda), r \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r), \\
&\qquad R_0 \leftarrow \mathsf{HSS.Eval}(0, \mathsf{ek}_0, \mathsf{k}_0, f \circ \mathsf{PRG}), R_1 := f(\mathsf{PRG}(r)) - R_0\} \\
\overset{c}{\approx} &\{(R_0, R_1) \mid r \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), R_0 \leftarrow \mathcal{R}^m, R_1 := f(\mathsf{PRG}(r)) - R_0\} \\
\overset{c}{\approx} &\{(R_0, R_1) \mid X \xleftarrow{\$} \mathcal{U}^n(\mathcal{R}), R_0 \leftarrow \mathcal{R}^m, R_1 := f(X) - R_0\}
\end{aligned}
$$

as required, where the first transition follow by correctness of HSS, the second by pseudorandomness of outputs of HSS and the last by pseudorandomness of PRG.

Security. Let $\sigma \in \{0,1\}$. We have

$$
\begin{aligned}
&\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \\
\overset{c}{\approx} &\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda), r \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r), \\
&\qquad R_{1-\sigma} \leftarrow \mathsf{HSS.Eval}(1-\sigma, \mathsf{ek}_{1-\sigma}, \mathsf{k}_{1-\sigma}, f \circ \mathsf{PRG}), R_\sigma := f(\mathsf{PRG}(r)) - R_{1-\sigma}\} \\
\overset{c}{\approx} &\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda), r \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), r' \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), \\
&\qquad (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r'), R_{1-\sigma} \leftarrow \mathsf{HSS.Eval}(1-\sigma, \mathsf{ek}_{1-\sigma}, \mathsf{k}_{1-\sigma}, f \circ \mathsf{PRG}), \\
&\qquad R_\sigma := f(\mathsf{PRG}(r)) - R_{1-\sigma}\} \\
\overset{c}{\approx} &\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda), X \xleftarrow{\$} \mathcal{U}^n(\mathcal{R}), r' \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), \\
&\qquad (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r'), R_{1-\sigma} \leftarrow \mathsf{HSS.Eval}(1-\sigma, \mathsf{ek}_{1-\sigma}, \mathsf{k}_{1-\sigma}, f \circ \mathsf{PRG}), \\
&\qquad R_\sigma := f(X) - R_{1-\sigma}\},
\end{aligned}
$$

where the first transition follows by correctness of HSS, the second transition by security of HSS and the last by pseudorandomness of PRG.

基于LPN的silent OT-extension：

这部分给出sublinear communication complexity生成n个random OT的protocol。

首先微调（BCGI18）的构造得到correlated OT，然后用（IKNP03）的OT extension protocol，得到一个random OT的PCG。然后用（Ds17）安全地生成seed，最终就有sublinear OT extension。

构造sVOLE的PCG：

subfield vector oblivious linear evaluation (sVOLE)定义：

Subfield VOLE is a form of vector oblivious linear evaluation (VOLE) over $\mathbb{F}_q$, which computes $\boldsymbol{w} = \boldsymbol{u}x + \boldsymbol{v}$, where the vector $\boldsymbol{u}$ is restricted to lie over a subfield $\mathbb{F}_p \subset \mathbb{F}_q$, for $q = p^r$ (and we multiply $\boldsymbol{u}$ with $x \in \mathbb{F}_q$ component-wise, by viewing x as a vector over $\mathbb{F}_p$). It outputs $(\boldsymbol{u}, \boldsymbol{v})$ to the sender and $(x, \boldsymbol{w})$ to the receiver.

记号说明：

The construction in Fig. 3 uses the function $\mathsf{spread}_n(S, \boldsymbol{y})$, which expands a set $S = (s_1, \ldots, s_{|S|}) \subset [n]$ and a vector $\boldsymbol{y} \in \mathbb{F}_p^{|S|}$ into the vector $\boldsymbol{\mu} \in \mathbb{F}_p^n$, where $\mu_{s_i} = y_i$ for $i = 1, \ldots, |S|$, and $\mu_j = 0$ for $j \in [n] \setminus S$. It is a generalization of the VOLE generator from [BCGI18], which follows from the case $p = q$.

具体构造：

Construction G_{sVOLE}

PARAMETERS:

- Security parameter 1^λ, integers $n' > n$, $q = p^r$, and noise weight t.
- A code generation algorithm $\mathbf{C}$ and $H_{n',n} \xleftarrow{\$} \mathbf{C}(n', n, \mathbb{F}_p)$.
- A multi-point FSS scheme $(\mathsf{MPFSS.Gen}, \mathsf{MPFSS.FullEval})$.

CORRELATION: Output $(\boldsymbol{u}, \boldsymbol{v})$ and $(x, \boldsymbol{w})$, where $x \leftarrow \mathbb{F}_q$, $\boldsymbol{u} \xleftarrow{\$} \mathbb{F}_p^n$, $\boldsymbol{v} \xleftarrow{\$} \mathbb{F}_q^n$ and $\boldsymbol{w} = \boldsymbol{u}x + \boldsymbol{v}$.

GEN: On input 1^λ:

1. Pick a random size-t subset S of $[n']$, sorted in increasing order.
2. Pick a random vector $\boldsymbol{y} \in (\mathbb{F}_p^*)^t$ and $x \xleftarrow{\$} \mathbb{F}_q$.
3. Compute $(K_0^{\mathsf{fss}}, K_1^{\mathsf{fss}}) \xleftarrow{\$} \mathsf{MPFSS.Gen}(1^\lambda, f_{S, x\cdot\boldsymbol{y}})$.
4. Let $\mathsf{k}_0 \leftarrow (m, n, K_0^{\mathsf{fss}}, S, \boldsymbol{y})$ and $\mathsf{k}_1 \leftarrow (m, n, K_1^{\mathsf{fss}}, x)$.
5. Output $(\mathsf{k}_0, \mathsf{k}_1)$.

EXPAND: On input $(\sigma, \mathsf{k}_\sigma)$:

1. If $\sigma = 0$: parse k_0 as $(m, n, K_0^{\mathsf{fss}}, S, \boldsymbol{y})$. Set $\boldsymbol{\mu} \leftarrow \mathsf{spread}_{n'}(S, \boldsymbol{y})$ in $\mathbb{F}_p^{n'}$. Compute $\boldsymbol{v}_0 \leftarrow \mathsf{MPFSS.FullEval}(0, K_0^{\mathsf{fss}})$ in $\mathbb{F}_q^{n'}$. Output $(\boldsymbol{u}, \boldsymbol{v}) \leftarrow (\boldsymbol{\mu} \cdot H_{n',n}, -\boldsymbol{v}_0 \cdot H_{n',n})$.
2. If $\sigma = 1$: parse k_1 as $(m, n, K_1^{\mathsf{fss}}, x)$. Compute $\boldsymbol{v}_1 \leftarrow \mathsf{MPFSS.FullEval}(1, K_1^{\mathsf{fss}})$ in $\mathbb{F}_q^{n'}$, and output $(x, \boldsymbol{w} \leftarrow \boldsymbol{v}_1 \cdot H_{n',n})$.

Fig. 3. PCG for subfield vector-OLE

定理陈述和证明：

Theorem 24. *Suppose the* $(\mathcal{HW}_t, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(n', n)$ *assumption holds, and that* MPFSS *is a secure multi-point FSS scheme. Then the construction* G_{sVOLE} *(Fig. 3) is a secure PCG for the subfield vector-OLE correlation.*

Proof. First let $\sigma = 0$. Here, in the real distribution, the adversary is given a key $\mathsf{k}_0 = (m, n, K_0^{\mathsf{fss}}, S, \boldsymbol{y})$, where $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} G_{\mathsf{sVOLE}}.\mathsf{Gen}(1^\lambda)$, as well as the expanded output $R_1 = (x, \boldsymbol{w}) \xleftarrow{\$} G_{\mathsf{sVOLE}}.\mathsf{Expand}(\mathsf{k}_1)$. We need to show that this is indistinguishable from the ideal distribution, where $R_1 \xleftarrow{\$} \mathsf{RSample}(0, R_0)$.

Recall that $\mathsf{RSample}(0, R_0)$ proceeds by sampling $x \xleftarrow{\$} \mathbb{F}_q$ and outputing $(x, \boldsymbol{w} = \boldsymbol{u}x + \boldsymbol{v})$. In the real distribution, x is also uniformly random, and from the correctness of MPFSS we have that

$$\boldsymbol{w} = \boldsymbol{v}_1 \cdot H_{n',n} = (\boldsymbol{v}_0 + x \cdot \mathsf{spread}_{n'}(S, \boldsymbol{y})) \cdot H_{n',n} = \boldsymbol{v} + x \cdot \boldsymbol{u}$$

which is identically distributed to the ideal distribution.

Next consider the case of $\sigma = 1$. We use the following sequence of games.

Game G_0**.** This is the real distribution, where the adversary gets $\mathsf{k}_1 = (m, n, K_1^{\mathsf{fss}}, x)$ and $R_0 = (\boldsymbol{u}, \boldsymbol{v}) \xleftarrow{\$} G_{\mathsf{sVOLE}}.\mathsf{Expand}(0, \mathsf{k}_0)$, that is, $\boldsymbol{u} = \boldsymbol{\mu} \cdot H_{n',n}$ for $\boldsymbol{\mu} \xleftarrow{\$} \mathcal{HW}_{t,n'}(\mathbb{F}_p)$ and $\boldsymbol{v} = \boldsymbol{v}_0 \cdot H_{n',n} = (\boldsymbol{v}_1 + x \cdot \boldsymbol{\mu}) \cdot H_{n',n}$.

Game G_1**.** Here, we compute K_1^{fss} using the MPFSS simulator $\mathsf{Sim}(1^\lambda, \cdots)$, and $\boldsymbol{v} = (\boldsymbol{v}_1 + x \cdot \boldsymbol{\mu}) \cdot H_{n',n}$. This is indistinguishable from G_0, by the security of the MPFSS.

Game G_2. Finally, here we compute $\boldsymbol{u} \xleftarrow{\$} \mathbb{F}_p^n$ instead of $\boldsymbol{\mu} \cdot H_{n',n}$, and let $\boldsymbol{v} = \boldsymbol{v}_1 \cdot H_{n',n} + x \cdot \boldsymbol{u}$. Notice that since K_1^{fss} is independent of $\boldsymbol{\mu}$, any adversary distinguishing G_1 and G_2 can be used to attack $(\mathcal{HW}_t, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(n', n)$.

Game G_2 is identical to the ideal distribution, so this completes the proof of the security property.

Finally, we show the correctness property, namely, that the outputs $(R_0, R_1) = (\boldsymbol{u}, \boldsymbol{v}, x, \boldsymbol{w})$ are computationally indistinguishable from outputs of $\mathcal{D}(1^\lambda)$. By the same reasoning as security for $\sigma = 0$, R_1 is already identically distributed to the output of $\mathsf{RSample}(0, R_0)$, so we write $\boldsymbol{w} = \boldsymbol{u}x + \boldsymbol{v}$. Denoting the uniform distribution on $\mathbb{F}_p^n$ by U_p^n, we then use the following sequence of hops:

$$
\begin{aligned}
(\boldsymbol{u}, \boldsymbol{v}, x, \boldsymbol{w}) &= (\boldsymbol{\mu} \cdot H_{n',n}, (\boldsymbol{v}_1 + x \cdot \boldsymbol{\mu}) \cdot H_{n',n}, x, \boldsymbol{u}x + \boldsymbol{v}) \\
&\overset{\mathrm{c}}{\approx} (\boldsymbol{\mu} \cdot H_{n',n}, (U_q^{n'} + x \cdot \boldsymbol{\mu}) \cdot H_{n',n}, x, \boldsymbol{u}x + \boldsymbol{v}) && (3) \\
&\overset{\mathrm{s}}{\approx} (\boldsymbol{\mu} \cdot H_{n',n}, U_q^n, x, \boldsymbol{u}x + \boldsymbol{v}) && (4) \\
&\overset{\mathrm{c}}{\approx} (U_p^n, U_q^n, x, \boldsymbol{u}x + \boldsymbol{v}) && (5) \\
&\equiv \mathcal{D}(1^\lambda)
\end{aligned}
$$

where (3) follows from the pseudorandomness of the MPFSS outputs (cf. Remark 2). Hop (4) holds because the LPN assumption implies from Remark 8 that $H_{n',n}$ must be full-rank with overwhelming probability, so preserves uniformity when multiplying by a uniform vector. Finally, (5) also holds due to the pseudorandomness of the LPN assumption.

Remark 2. In any FSS scheme for a sufficiently rich class of functions (including point functions), each of the two evaluation functions $F_K^b(x) = \mathsf{FSS.Eval}(b, K, x)$ is a pseudorandom function [BGI15]. Some of our constructions will use this property.

根据subfield VOLE可以构造correlated OT（Δ-OT），就是一种sender的数组两个数差值为定值的OT，在很多practical MPC中有实际的用处。事实上，取域为二元域， 再交换sender和receiver的地位，就可以得到。

还可以用于构造矩阵乘法，把乘法作为tensor product展开成分量就是矩阵形式。

Subfield VOLE immediately gives a PCG for *correlated OT* (or Δ-OT). This is a batch of 1-out-of-2 OTs where the sender's strings are of the form $(w_i, w_i \oplus \Delta)$ for some fixed string Δ, and is the main building block in practical MPC protocols such as TinyOT [NNOB12] and authenticated garbling [WRK17a, WRK17b].

To obtain correlated OT, we run subfield VOLE with $p = 2$ and $q = 2^r$, so the VOLE sender obtains $u_i \in \mathbb{F}_2, v_i \in \mathbb{F}_{2^r}$, while the VOLE receiver gets $x \in \mathbb{F}_{2^r}$ and $w_i = x \cdot u_i + v_i$, for $i = 1, \ldots, n$. Now switching the roles of sender and receiver, the VOLE sender can be seen as an OT receiver with choice bit u_i and string v_i. This gives us a correlated OT, since the OT sender (formerly VOLE receiver) can compute the strings $(w_i, w_i + x)$, and we have $v_i = w_i$ if $u_i = 0$ and $v_i = w_i + x$ if $u_i = 1$.

PCG for random OT:

上面的sVOLE可以给出correlated OT，我们使用（IKNP03）的OT extension技术，使用特别的hash函数可以将correlated OT转变为random OT。

使用的一种重要工具：Correlation Robust Function：

在[GKWY19]中, 他可以用 fixed-key AES modeled as a random permutation 来实现当p = 2时。

Definition 25 ($\mathbb{F}_p$-correlation robust function). *Let $n = \mathsf{poly}(\lambda)$ and $t_1, \ldots, t_n, x$ be uniformly sampled from $\mathbb{F}_p^r$, where $p^r = \lambda^{\omega(1)}$. Then, $\mathsf{H} : \{0,1\}^\lambda \times \mathbb{F}_p^r \to \{0,1\}^\lambda$ is* $\mathbb{F}_p$-correlation robust *if the distribution*

$$\left(t_1, \ldots, t_n, \{\mathsf{H}(1, t_1 - j \cdot x), \ldots, \mathsf{H}(n, t_n - j \cdot x)\}_{j \in \mathbb{F}_p \setminus \{0\}}\right)$$

is computationally indistinguishable from uniform on $\mathbb{F}_p^{rn} \times \{0,1\}^{\lambda(p-1)n}$.

主要的构造：

Construction G_{OT}

PARAMETERS:

- Security parameter 1^λ, integers n, $q = p^r = \lambda^{\omega(1)}$.
- An $\mathbb{F}_p$-correlation-robust function $\mathsf{H} : \{0,1\}^\lambda \times \mathbb{F}_q \to \{0,1\}^\lambda$.
- The subfield-VOLE PCG $(G_{\mathsf{sVOLE}}.\mathsf{Gen}, G_{\mathsf{sVOLE}}.\mathsf{Expand})$

CORRELATION: Outputs $(R_0, R_1) = \left(\{(u_i, w_{i,u_i})\}_{i \in [n]}, \{w_{i,j}\}_{i \in [n], j \in [p]}\right)$, where $w_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$ and $u_i \xleftarrow{\$} \{1, \ldots, p\}$, for $i \in [n], j \in [p]$.

GEN: On input 1^λ, output $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow G_{\mathsf{sVOLE}}.\mathsf{Gen}(1^\lambda, n, p, q)$.

EXPAND: On input $(\sigma, \mathsf{k}_\sigma)$:

1. If $\sigma = 0$: compute $(\boldsymbol{u}, \boldsymbol{v}') \leftarrow G_{\mathsf{sVOLE}}.\mathsf{Expand}(\sigma, \mathsf{k}_\sigma)$, where $\boldsymbol{u} \in \mathbb{F}_p^n, \boldsymbol{v}' \in \mathbb{F}_q^n$. Compute
$$v_i \leftarrow \mathsf{H}(i, v_i') \quad \text{for } i = 1, \ldots, n$$
and output (u_i, v_i).
2. If $\sigma = 1$: compute $(x, \boldsymbol{w}') \leftarrow G_{\mathsf{sVOLE}}.\mathsf{Expand}(\sigma, \mathsf{k}_\sigma)$, where $x \in \mathbb{F}_q, \boldsymbol{w}' \in \mathbb{F}_q^n$. Compute
$$w_{i,j} \leftarrow \mathsf{H}(i, w_i' - j \cdot x) \quad \text{for } i = 1, \ldots, n, \forall j \in \mathbb{F}_p$$
and output $\{w_{i,j}\}_{i,j}$.

Fig. 4. PCG for n sets of 1-out-of-p random OT

主要的定理和证明：

Theorem 26. *Suppose that* H *is an* $\mathbb{F}_p$*-correlation robust hash function and* G_{sVOLE} *is a secure PCG. Then the silent OT construction (Fig. 4) is a secure PCG for the random 1-out-of-p OT correlation.*

Proof. We start by showing the correctness property. First, from the correctness of G_{sVOLE} we have

$$v_i = \mathsf{H}(i, v'_i) = \mathsf{H}(i, w'_i - u_i \cdot x) = w_{i,u_i}$$

as required. Indistinguishability of the outputs from OT_p^n follows first from indistinguishability of the VOLE outputs (u_i, v'_i, x, w'_i), and secondly by a standard reduction to the $\mathbb{F}_p$-correlation robustness property of H.

$$u_i, v_i, w_{i,j} = u_i, \mathsf{H}(i, w'_i - u_i \cdot x) \overset{\text{c}}{\approx} (U, \mathsf{H}(i,))$$

We now consider the security property, for the case $\sigma = 0$. Here, the real distribution consists of the seed k_0 and the sender's outputs $w_{i,j}$, for $i = 1, \ldots, n$ and $j \in \mathbb{F}_p$. From correctness we have

that for $\boldsymbol{u}, \boldsymbol{v}' \leftarrow G_{\mathsf{sVOLE}}.\mathsf{Expand}(0, \mathsf{k}_0)$, it holds that $\mathsf{H}(i, v'_i) = w_{i,u_i}$. From the security property of G_{sVOLE}, we can replace all the sender's outputs $w_{i,j}$, except for w_{i,u_i}, with ones computed using uniform values x, w'_i, instead of from $G_{\mathsf{sVOLE}}.\mathsf{Expand}$. These are then indistinguishable from uniform under the $\mathbb{F}_p$-correlation robustness of H.

When $\sigma = 1$, the real distribution contains the seed k_1 and the receiver's outputs u_i, v_i. As before, from the correctness property we have that $v_i = w_{i,u_i}$, where $w_{i,j}$ is computed from k_1 via $G_{\mathsf{sVOLE}}.\mathsf{Expand}$ and the hash function. We only need to show that u_i is uniform, which follows directly from the security property of G_{sVOLE} when $\sigma = 1$.

最后用（Ds17）的distributed point function，得到random OT的整体实现，并分析复杂性：

To do this efficiently with semi-honest security, we use the black-box protocol of Doerner and shelat [Ds17] (also used in [BCGI18]) for setting up distributed point function keys. For a single point function of domain size n, this requires $O(\log n)$ OTs on $O(\lambda)$-bit strings, giving $O(t \log n)$ OTs for a multi-bit point function. Implementing each OT with (non-silent) OT extension [IKNP03] costs $O(\lambda)$ bits of communication, plus a setup phase of λ base OTs. Putting this together, we obtain the following.

Theorem 27. *Suppose the* $(\mathcal{HW}_t, \mathbf{C}, \mathbb{F}_p)$-$\mathsf{dual\text{-}LPN}(n', n)$ *assumption holds, and an* $\mathbb{F}_p$*-correlation robust hash function exists. Then there is a protocol that uses* $O(\lambda)$ *1-out-of-2 OTs to realize* n *instances of random 1-out-of-p OT with semi-honest security, using* $O(t\lambda \log n) + \mathsf{poly}(\lambda)$ *bits of communication.*

We remark that this gives OT with *sublinear communication* when $t = o(n/(\lambda \log n))$, which translates to an instance of LPN with noise rate $1/\omega(\lambda \log n)$. If the matrix $H_{n',n}$ in G_{sVOLE} is uniformly random, the computational complexity is dominated by $O(n' \cdot n)$ arithmetic operations; using more structured matrices based on LDPC codes or quasi-cyclic codes, we get respective costs of $O(n')$ or $\tilde{O}(n')$ arithmetic and PRG operations.

PCG for Constant-Degree Correlations from LPN：

先给出一个Bilinear correlation的构造（是constant-degree correlation的一种特例）

> More precisely, we consider the following type of additive correlations: the party P_σ receives pseudorandom vectors (x_σ, z_σ) such that $B(x_0, x_1) = z_0 + z_1$, where B is a bilinear function.

事实上，每个人掌握一个input和把input share给两个人是等价的：

> the general case (where the inputs are shared as well) can be obtained in a blackbox way using two parallel calls to the PCG for these restricted forms of bilinear correlations

We demonstrate this with the construction below, where BCG_s denotes a pseudorandom correlation generator for general bilinear correlations, and BCG denotes a PCG for restricted bilinear correlations. 黑盒证明如下：

- $\mathsf{BCG}_s.\mathsf{Setup}(1^\lambda)$ outputs $(\mathsf{pp}, \mathsf{sk}) \xleftarrow{\$} \mathsf{BCG.Setup}(1^\lambda)$;
- $\mathsf{BCG}_s.\mathsf{Gen}(\mathsf{sk}, B)$ runs twice $\mathsf{BCG.Gen}(\mathsf{sk}, B)$, and outputs $((\mathsf{k}_0, \mathsf{k}'_0), (\mathsf{k}_1, \mathsf{k}'_1))$;
- $\mathsf{BCG}_s.\mathsf{Expand}(\mathsf{pp}, \sigma, (\mathsf{k}_\sigma, \mathsf{k}'_\sigma))$ runs twice $\mathsf{BCG.Expand}$, and outputs $\boldsymbol{z}_\sigma = ((1-\sigma)\boldsymbol{x}_\sigma + \sigma\boldsymbol{x}'_\sigma, \sigma\boldsymbol{x}_\sigma + (1-\sigma)\boldsymbol{x}'_\sigma, \boldsymbol{y}_\sigma + \boldsymbol{y}'_\sigma + B(\boldsymbol{x}_\sigma, \boldsymbol{x}'_\sigma))$.

Then, it holds that

$$\begin{aligned}\boldsymbol{z}_0 + \boldsymbol{z}_1 &= (\boldsymbol{x}_0 + \boldsymbol{x}'_1, \boldsymbol{x}'_0 + \boldsymbol{x}_1, B(\boldsymbol{x}_0, \boldsymbol{x}_1) + B(\boldsymbol{x}'_0, \boldsymbol{x}'_1) + B(\boldsymbol{x}_0, \boldsymbol{x}'_0) + B(\boldsymbol{x}_1, \boldsymbol{x}'_1)) \\ &= (\boldsymbol{x}_0 + \boldsymbol{x}'_1, \boldsymbol{x}'_0 + \boldsymbol{x}_1, B(\boldsymbol{x}_0 + \boldsymbol{x}'_1, \boldsymbol{x}'_0 + \boldsymbol{x}_1)) \\ &= (\boldsymbol{x}, \boldsymbol{x}', B(\boldsymbol{x}, \boldsymbol{x}')), \text{ denoting } \boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{x}'_1, \boldsymbol{x}' = \boldsymbol{x}'_0 + \boldsymbol{x}_1.\end{aligned}$$

总体思路：（from BCG+20）

Construction from [BCG+19b]. Before describing our PCG for OLE, it is instructive to recall the PCG for general degree-two correlations by Boyle et al. [BCG+19b], based on LPN. The goal is to build a PCG for the correlation which gives each party a random vector $\vec{x}_i$, together with an additive secret share of the tensor product $\vec{x}_0 \otimes \vec{x}_1$. They used the dual form of LPN over a ring $\mathbb{Z}_p$, which states that the distribution

$$\left\{ H, H \cdot \vec{e} \,\middle|\, H \xleftarrow{\$} \mathbb{Z}_p^{m \times n}, \vec{e} \xleftarrow{\$} \mathbb{Z}_p^n \text{ s.t. } \mathrm{wt}(\vec{e}) = t \right\}$$

is computationally indistinguishable from uniform, where $\vec{e}$ is a sparse random vector with only t non-zero coordinates, for some $t \ll n$, and $m < n$.

The idea of the construction is that the setup algorithm gives each party a random sparse $\vec{e}_0$ or $\vec{e}_1$, and computes the tensor product $\vec{e}_0 \otimes \vec{e}_1$, which has at most t^2 non-zero coordinates. This product is then distributed to the parties via function secret sharing (FSS), by generating a pair of FSS keys for the function that outputs each entry of the product on its respective inputs from 1 to n^2. This function can be written as a sum of t^2 point functions, allowing practical FSS schemes based on distributed point functions [GI14, BGI15, BGI16b]. Note that unlike the

case of PCGs for OT or Vector-OLE [BCG+19a, SGRR19], here we cannot replace FSS by the simpler punctured PRF primitive.

Given shares of $\vec{e}_0 \otimes \vec{e}_1$ and either $\vec{e}_0$ or $\vec{e}_1$, the parties expand these using LPN, computing:

$$\vec{x}_0 = H \cdot \vec{e}_0, \quad \vec{x}_1 = H \cdot \vec{e}_1, \quad \vec{z} = (H \cdot \vec{e}_0) \otimes (H \cdot \vec{e}_1) = (H \otimes H) \cdot (\vec{e}_0 \otimes \vec{e}_1)$$

where $\vec{x}_i$ is computed by party P_i, while $\vec{z}$ is computed in secret-shared form, using the shares of $\vec{e}_0 \otimes \vec{e}_1$ and the formula on the right-hand side.

Notice that both $\vec{x}_0$ and $\vec{x}_1$ are pseudorandom under LPN, which gives the desired correlation.

Bilinear Correlation的构造：

Construction G_{bil}

PARAMETERS: $1^\lambda, n, n', t, p \in \mathbb{N}$, where $n' > n$. A code generation algorithm $\mathbf{C}$ and $H_{n',n} \xleftarrow{\$} \mathbf{C}(n', n, \mathbb{F}_p)$. A bilinear function $B_{\boldsymbol{c}} : (\boldsymbol{\alpha}, \boldsymbol{\beta}) \to \boldsymbol{c} \cdot (\boldsymbol{\alpha} \otimes \boldsymbol{\beta})^\intercal$, where $\otimes$ denotes the tensor product.

Gen: On input 1^λ:

1. Pick two random size-t subsets (S_0, S_1) of $[n']$, sorted in increasing order.
2. Pick two random vector $(\boldsymbol{y}_0, \boldsymbol{y}_1) \in (\mathbb{F}_p^t)^2$.
3. Compute $(K_0^{\mathsf{fss}}, K_1^{\mathsf{fss}}) \xleftarrow{\$} \mathsf{MPFSS.Gen}(1^\lambda, f_{S_0 \times S_1, \boldsymbol{y}_0 \otimes \boldsymbol{y}_1})$.
4. Let $\mathsf{k}_0 \leftarrow (n, K_0^{\mathsf{fss}}, S_0, \boldsymbol{y}_0)$ and $\mathsf{k}_1 \leftarrow (n, K_1^{\mathsf{fss}}, S_1, \boldsymbol{y}_1)$.
5. Output $(\mathsf{k}_0, \mathsf{k}_1)$.

Expand: On input $(\sigma, \mathsf{k}_\sigma)$, parse k_σ as $(n, K_\sigma^{\mathsf{fss}}, S_\sigma, \boldsymbol{y}_\sigma)$. Set $\boldsymbol{\mu}_\sigma \leftarrow \mathsf{spread}_{n'}(S_\sigma, \boldsymbol{y}_\sigma)$ in $\mathbb{F}_p^{n'}$ and $\boldsymbol{x}_\sigma \leftarrow \boldsymbol{\mu}_\sigma \cdot H_{n',n}$. Compute $\boldsymbol{v}_\sigma \leftarrow \mathsf{MPFSS.FullEval}(\sigma, K_\sigma^{\mathsf{fss}})$ in $\mathbb{F}_p^{(n')^2}$ and set $\boldsymbol{z}_\sigma \leftarrow -\boldsymbol{c} \cdot (\boldsymbol{v}_\sigma \cdot (H_{n',n} \otimes H_{n',n}))^\intercal$. Output $(\boldsymbol{x}_\sigma, \boldsymbol{z}_\sigma)$.

Theorem 28. *Suppose the $(\mathcal{HW}_t, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(n', n)$ assumption holds, and that* MPFSS *is a secure multi-point FSS scheme. Then the construction G_{bil} (Fig. 5) is a secure PCG for general bilinear correlations.*

效率分析：

Efficiency. Instantiating the MPFSS as in [BCGI18], the setup algorithm of G_{bil} outputs seeds of size $t^2 \cdot (\lceil \log n' \rceil (\lambda + 2) + \lambda + \log_2 |\mathbb{F}|)$ bits, which amounts to $\tilde{O}(t^2)$ field elements over a large field ($\log_2 |\mathbb{F}| = O(\lambda)$). Expanding the seed involves $(tn')^2$ PRG evaluations and $O(n \cdot n')^2 = O(n^4)$ arithmetic operations.

效率不高的主要问题和优化方法： （from BCG+20）

Optimizations and additional applications. Boyle et al. state the computational complexity of the above as $O(n^4)$ operations, due to the tensor product of H with itself. We observe that the value of $(H \cdot \vec{e}_0) \otimes (H \cdot \vec{e}_1)$ can be read directly from $H \cdot (\vec{e}_0 \cdot \vec{e}_1^{\intercal}) \cdot H^{\intercal}$, which requires much less computation and can be made even more efficient if H is a structured matrix, reducing the computational complexity to $\tilde{O}(n^2)$. We also describe two variants of the PCG which allow producing large matrix multiplication correlations with different parameter tradeoffs. While much less practical than our main constructions, we present these in Section 10 for completeness.

A first attempt. The problem with the above construction is that it produces an entire tensor product correlation, which inherently requires $\Omega(n^2)$ computation. Even if we only want to compute the diagonal entries of the tensor product output (that is, n OLEs), we do not see a way to do this any more efficiently.

The bottleneck is computing the n^2 entries of $\vec{e}_0 \otimes \vec{e}_1$ to obtain $\vec{z}$. A natural idea to reduce the computational complexity is to replace $\vec{e}_0$ and $\vec{e}_1$ by degree-n sparse polynomials e_0 and e_1, and $\vec{x}_0$ and $\vec{x}_1$ by, say, $n/2$ evaluations of e_0 and e_1 respectively. Then, $\vec{z}$ is computable in quasilinear time as evaluations of the polynomial product $e_0 \cdot e_1$ of degree $2n$. This approach does not give a secure PCG candidate, though, because $\vec{x}_0$ and $\vec{x}_1$ can be efficiently distinguished from random using algebraic decoding techniques.

推广到任意次数的Constant-Degree Polynomial Correlations：

通过把share对象改为高维的tensor product，可以同样的获得高次多项式的share。具体地：

Generalization. The scheme G_{bil} immediately generalizes to a PCG for arbitrary constant-degree polynomials,[13] where the size of the shares grows as $\tilde{O}(t^d)$ and the computational complexity is $\tilde{O}((tn')^d + (nn')^d)$. It allows two parties to locally compute, given the shares, additive shares of $(\boldsymbol{r}, P(\boldsymbol{r}))$, where $\boldsymbol{r}$ is pseudorandom (under LPN) and P is a degree-d multivariate polynomial over $\mathbb{F}$.

To see this, notice that we can replace $\boldsymbol{y}_0 \otimes \boldsymbol{y}_1$ in **Gen** with $\otimes_d \boldsymbol{y} = \boldsymbol{y} \otimes \cdots \otimes \boldsymbol{y}$, where $\otimes_d \boldsymbol{y}$ denotes the tensor product of $\boldsymbol{y}$ with itself d times (that is, the list of all degree-d monomials of $\boldsymbol{y}$). The parties can then compute shares of all degree-d terms in $P(\boldsymbol{r})$ for a random $\boldsymbol{r}$; to obtain shares of $\boldsymbol{r}$ and the lower-degree terms, we extend the MPFSS values to include $(\boldsymbol{y}, \boldsymbol{y} \otimes \boldsymbol{y}, \cdots, \otimes_{d-1} \boldsymbol{y})$ as well as $\otimes_d \boldsymbol{y}$.

总结一下就是：

Corollary 29. *Suppose the* $(\mathcal{HW}_t, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(n', n)$ *assumption holds, and that* MPFSS *is a secure multi-point FSS scheme. Then there exists a secure PCG for general constant-degree correlations, with share size* $\tilde{O}(t^d)$ *and computational complexity* $O((n \cdot n')^d)$.

基于Groups和Lattices的PCG构造：

这一部分的出发点是section 4.4的基于HSS构造PCG。

我们要给出好几种CR的构造，具体地说，PCGs for the generation of *bilinear correlations* from groups, and PCGs for so-called *authenticated Beaver triples* from lattices.

首先给出两个我们要用的好的PRG：

基于MQ：

Remark 34 (PRG from MQ). Let $\mathcal{R}$ be a ring, and $\ell, n \in \mathbb{N}$. Let $\mathcal{M}$ be a distribution over $\mathcal{R}^{\ell^2 \times n}$ and $\mathbf{M} \xleftarrow{\$} \mathcal{M}(\ell^2, n, \mathcal{R})$. We assume that for an appropriate choice of parameters

$$\mathsf{PRG}_{\mathsf{MQ}} \colon \mathcal{R}^{\ell} \to \mathcal{R}^{n}, r \mapsto \mathbf{M}^{\top} \cdot (r \otimes r)$$

is a PRG. We say $\mathcal{M}(\ell^2, n, \mathcal{R})$ has *sparsity* ρ, if for every matrix $\mathbf{M}$ in the image of $\mathcal{M}(\ell^2, n, \mathcal{R})$, the number of non-zero entries in any column of $\mathbf{M}$ is at most ρ.

tip：MQ被认为是一个比较安全的assumption，（in particular, the pseudorandomness of the MQ-based PRG reduces to the conjectured one-wayness of solving a random system of quadratic assumptions [BGP06]），尽管多变量公钥体制经常被构造然后攻破，但是似乎现有的攻击表明对多变量加密体制的攻击并不规约到对MQ的攻击。

基于LPN：

Remark 35 ($\mathcal{D}^{\ell}(\mathcal{R})$-PRG from LPN). Let $\mathcal{R}$ be a ring, and $\ell, k, c, \tau, n \in \mathbb{N}$, such that $\ell = \tau c k$. Let $\mathcal{M}$ be a distribution on $\mathcal{R}^{\tau^c \times n}$ and $\mathbf{M} \xleftarrow{\$} \mathcal{M}(\tau^c, n, \mathcal{R})$. Let

$$\mathsf{PRG}_{\mathsf{LPN}} \colon (\mathcal{R}^{\tau})^{ck} \to \mathcal{R}^{n}, (r_1, \ldots, r_{ck}) \mapsto \mathbf{M}^{\top} \cdot \sum_{i=0}^{k-1} r_{1+i\cdot c} \otimes \cdots \otimes r_{c+i\cdot c},$$

where $r_i \in \{0,1\}^{\tau}$ for all $1 \le i \le ck$. We assume that for appropriate choice of parameters $\mathsf{PRG}_{\mathsf{LPN}}$ is a $\mathcal{D}^{\ell}(\mathcal{R})$-PRG, where $\mathcal{D}^{\ell}$ is the distribution returning vectors that have *exactly* one non-zero entry (chosen uniformly at random in $\mathcal{R}\backslash\{0\}$) in every block of length τ.

Note that $\sum_{i=0}^{k-1} r_{1+i\cdot c} \otimes \cdots \otimes r_{c+i\cdot c}$ yields a random vector in $\mathcal{R}^{\tau^c}$ with exactly k non-zero entries. Therefore, the above assumption corresponds to the $(\mathsf{Ber}_k(\mathcal{R})^{\tau^c}, \mathcal{M}, \mathcal{R})$-dual-LPN$(\tau^c, n)$ assumption, where $\mathsf{Ber}_k(\mathcal{R})^{\tau^c}$ is the distribution returning vectors in $\mathcal{R}^{\tau^c}$ with exactly k non-zero entries.

Group-Based PCG for Bilinear Correlations

方法概述：

由于group-based都是有错误概率的，所以我们构造一种有错误概率的新型的PCG：Sanitizable Bilinear Correlation Generator，再定义一种新型的HSS，也是有错误概率的：Las Vegas Homomorphic Secret Sharing，还有一种share的长度有上界的HSS：$(\mathcal{D}, comp)$-Compressible HSS。

然后我们可以构造BCG（Bilinear Correlation Generator）from LPN and Degree-2 CHSS：

Theorem 47. *Assuming the $(\mathcal{D}(\mathcal{R}), \mathbf{C})$-LPN$(\ell, \ell - n)$ assumption, the above construction is a bilinear correlation generator with seed size upper-bounded by $2 \cdot \mathsf{comp}(\lambda, \ell)$.*

还可以构造Sanitizable BCG from LPN and Compressible Las Vegas HSS.
现在就差给出其中的CHSS了。

下面我们考虑其中的HSS的构造：Group-Based HSS for Bilinear Functions。
以及ElGamal cryptosystem over composite-order pairing-friendly elliptic curve的一个变种：BGN-EG Cryptosystem。
接下来我们可以从BGN-EG构造Compressible HSS：我们说group-based HSS可以改造成compressible HSS。（这一步大概是这个section的核心）
具体地说我们得到：

Theorem 48. *Assuming the DDH assumption over pairing-friendly elliptic curves, for any integer t of polynomial size and any integer k, there exists a $(\mathsf{Ber}_{k/\ell}(\mathbb{Z}_t), \mathsf{comp}(\lambda, \ell, k))$-compressible (Las Vegas, secret-key, degree-2) CHSS, with*

$$\mathsf{comp}(\lambda, \ell, k) = k \cdot \sqrt{\ell} \cdot \mathsf{poly}(\lambda).$$

综上所述我们得到最终：

Theorem 49. *Assuming the DDH assumption over pairing-friendly elliptic curves and the $\mathsf{LPN}((\alpha-1)\cdot n, \alpha \cdot n, k/(\alpha n))$ assumption over an integer ring $\mathbb{Z}_t$ of polynomial size, for a positive constant $\alpha > 1$, there exists a δ-failure SBCG with seed size $k \cdot (\alpha \cdot n)^{1/2} \cdot \mathsf{poly}(\lambda)$.*

定义 Sanitizable Bilinear Correlation Generator：

直观：除了多项式倒数概率，correctness保持，并且parties可以检测何时一个输出大概率不正确，知道错误输出的位置后可以在sanitization phase中用一个比较高效的算法清除这个错误，用某种方式(such as punctured OT [BGI17] or leakage-absorbing pads [BCG+17])。
其中setup步骤主要是生成seed中可重复使用的部分(包括public and secret parameters)。

Definition 44 (Sanitizable Bilinear Correlation Generator). *A (δ-failure) sanitizable bilinear correlation generator over a ring $\mathcal{R}$ is a triple of algorithms* $(\mathsf{BCG.Setup}, \mathsf{BCG.Gen}, \mathsf{BCG.Expand})$ *with the following syntax:*

- $\mathsf{BCG.Setup}(1^\lambda)$ *is a PPT algorithm that given a security parameter λ, outputs public parameters* pp *and a secret key* sk*;*
- $\mathsf{BCG.Gen}(\mathsf{sk}, B)$ *is a PPT algorithm that given secret key* sk *and a bilinear map $B : \mathcal{R}^n \times \mathcal{R}^n \mapsto \mathcal{R}^m$, outputs a pair of seeds $(\mathsf{k}_0, \mathsf{k}_1)$;*
- $\mathsf{BCG.Expand}(\mathsf{pp}, \sigma, \mathsf{k}_\sigma, \delta)$ *is an algorithm running in time polynomial in λ and $1/\delta$ that, given party index $\sigma \in \{0,1\}$, a seed k_σ, and a failure bound δ, outputs a pair of vectors $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{R}^n \times \mathcal{R}^m$, as well as a list of m confidence flags $\gamma_{\sigma,i} \in \{\bot, \top\}$ for $i = 1$ to m to indicate full confidence ($\top$) or a possibility of failure ($\bot$) for any given output.*

The algorithms $(\mathsf{BCG.Setup}, \mathsf{BCG.Gen}, \mathsf{BCG.Expand})$ *should satisfy the following:*

- **δ-Correctness.** *For every $i \le m$ and every polynomial p, there is a negligible ν such that for every positive integer λ, bilinear function $B : \mathcal{R}^n \times \mathcal{R}^n \mapsto \mathcal{R}^m$, and failure bound $\delta > 0$, where $|B|, 1/\delta \le p(\lambda)$, we have:*

$$\Pr[(\gamma_{0,i} = \bot) \wedge (\gamma_{1,i} = \bot)] \le \delta + \nu(\lambda),$$

and

$$\Pr[((\gamma_{0,i} = \top) \vee (\gamma_{1,i} = \top)) \wedge y_{0,i} + y_{1,i} \neq B(\boldsymbol{x}_0, \boldsymbol{x}_1)_i] \le \nu(\lambda),$$

where the probability is taken over

$$(\mathsf{pp}, \mathsf{sk}) \xleftarrow{\$} \mathsf{BCG.Setup}(1^\lambda), (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{BCG.Gen}(\mathsf{sk}, B)$$

and where we denote $(\boldsymbol{x}_0, \boldsymbol{y}_0) \leftarrow \mathsf{BCG.Expand}(0, \mathsf{k}_0)$, and $(\boldsymbol{x}_1, \boldsymbol{y}_1) \leftarrow \mathsf{BCG.Expand}(1, \mathsf{k}_1)$.

- **Security.** *For any $\sigma \in \{0,1\}$ and any (stateful, nonuniform) polynomial-time adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{array}{ll}(\mathsf{pp}, \mathsf{sk}) \xleftarrow{\$} \mathsf{BCG.Setup}(1^\lambda), B \leftarrow \mathcal{A}(\mathsf{pp}), & \\ (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{BCG.Gen}(\mathsf{sk}, B), & : \mathcal{A}(\boldsymbol{x}_\sigma, \mathsf{k}_{1-\sigma}) = 1 \\ (\boldsymbol{x}_\sigma, \boldsymbol{y}_\sigma, \boldsymbol{\gamma}_\sigma) \leftarrow \mathsf{BCG.Expand}(\mathsf{pp}, \sigma, \mathsf{k}_\sigma) & \end{array}\right]$$

$$\approx \Pr\left[\begin{array}{ll}(\mathsf{pp}, \mathsf{sk}) \xleftarrow{\$} \mathsf{BCG.Setup}(1^\lambda), B \leftarrow \mathcal{A}(\mathsf{pp}), & \\ (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{BCG.Gen}(\mathsf{sk}, B), & : \mathcal{A}(\boldsymbol{x}_\sigma, \mathsf{k}_{1-\sigma}) = 1 \\ \boldsymbol{x}_\sigma \xleftarrow{\$} \mathcal{R}^n, \boldsymbol{\gamma}_\sigma \xleftarrow{\$} \mathsf{Ber}_\delta(\{\bot, \top\})^m & \end{array}\right].$$

定义 Las Vegas HSS：

就是有错误概率的HSS。

Definition 45 (Las Vegas Homomorphic Secret Sharing). *A (2-party, secret-key, Las Vegas δ-failure)* Degree-d Homomorphic Secret Sharing (HSS) *scheme over a ring $(\mathcal{R}, +, \cdot)$ is a triple of PPT algorithms* $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *with the following syntax:*

- $\mathsf{HSS.Gen}(1^\lambda)$*: On input a security parameter 1^λ, the key generation algorithm outputs a secret key* sk *and an evaluation key* ek*.*
- $\mathsf{HSS.Share}(\mathsf{sk}, x)$*: Given secret key* sk *and secret input value $x \in \mathcal{R}^n$, the sharing algorithm outputs a pair of shares (s_0, s_1). We assume that the input length n is included in each of (s_0, s_1).*
- $\mathsf{HSS.Eval}(\sigma, \mathsf{ek}, s_b, P, \delta)$*: On input party index $\sigma \in \{0, 1\}$, evaluation key ek_σ, share s_σ of a size-n input, degree-d arithmetic circuit P with n input bits and m output bits, and a failure bound δ, the homomorphic evaluation algorithm outputs $y_b \in \mathcal{R}^m$, constituting party b's share over $\mathcal{R}$ of an output $y \in \mathcal{R}^m$, as well as a confidence flag $\gamma_b \in \{\bot, \top\}$ to indicate full confidence ($\top$) or a possibility of failure ($\bot$).*

The algorithms $(\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *should satisfy the following correctness and security requirements:*

- **Correctness:**
  For every polynomial p there is a negligible ν such that for every sufficiently large integer λ, input $\boldsymbol{x} \in \mathcal{R}^n$, degree-$d$ arithmetic circuit P with input length n, and failure bound $\delta > 0$, where $|P|, 1/\delta \leq p(\lambda)$, we have:

$$\Pr[(\gamma_0 = \bot) \wedge (\gamma_1 = \bot)] \leq \delta + \nu(\lambda),$$

  and

$$\Pr[((\gamma_0 = \top) \vee (\gamma_1 = \top)) \wedge \boldsymbol{y}_0 + \boldsymbol{y}_1 \neq P(\boldsymbol{x})] \leq \nu(\lambda),$$

  where probability is taken over

$$(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{HSS.Gen}(1^\lambda); (s_0, s_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, \boldsymbol{x});$$
$$(\boldsymbol{y}_\sigma, \gamma_\sigma) \leftarrow \mathsf{HSS.Eval}(\sigma, \mathsf{ek}, s_\sigma, P, \delta),\ \sigma \in \{0, 1\}.$$

- **Security:** *For any $\sigma \in \{0, 1\}$, any pair of inputs x, x' of the same length, the distribution ensembles $C_\sigma(\lambda, x)$ and $C_\sigma(\lambda, x')$ are computationally indistinguishable, where $C_\sigma(\lambda, y)$ for $y \in \{x, x'\}$ is obtained by sampling $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{HSS.Gen}(1^\lambda)$, sampling $(s_0, s_1) \leftarrow \mathsf{HSS.Enc}(\mathsf{sk}, y)$, and outputting (ek, s_σ).*

定义 compressible HSS for a family of distributions D (CHSS)：

输入是从一个分布D中来的，并且share的大小 is upper-bounded by comp(λ,n), comp 称为 compression ratio。

Definition 46 ($(\mathcal{D}, \mathsf{comp})$-Compressible HSS). *A (2-party, secret-key, degree-d) compressible homomorphic secret sharing over a ring $\mathcal{R}$ for a family of distributions $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_n(\mathcal{R})\}_{n\in\mathbb{N}}$ (such that $\mathsf{Im}(\mathcal{D}_n(\mathcal{R})) \subseteq \mathcal{R}^n$) with compression ratio comp, or $(\mathcal{D}(\mathcal{R}), \mathsf{comp})$-CHSS, is a (2-party, secret-key, degree-d) homomorphic secret sharing over $\mathcal{R}$ whose correctness is relaxed as follows:*

- **Relaxed Correctness.** *For every sufficiently large positive integers λ, n and degree-d arithmetic circuit P with input length n, we have:*

$$\Pr[\boldsymbol{y}_0 + \boldsymbol{y}_1 \neq P(\boldsymbol{x})] \leq \mathsf{negl}(\lambda),$$

where probability is taken over

$$\begin{gathered}(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{HSS.Gen}(1^\lambda);\ \boldsymbol{x} \leftarrow \mathcal{D}(\mathcal{R})_n;\ (s_0, s_1) \xleftarrow{\$} \mathsf{HSS.Share}(\mathsf{sk}, \boldsymbol{x});\\ \boldsymbol{y}_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{ek}, s_b, P),\ b \in \{0,1\},\end{gathered}$$

and which satisfies an additional compressibility *property:*

- **Compressibility.** *A $(\mathcal{D}, \mathsf{comp})$-CHSS is* compressible with compression ratio comp *if for every sufficiently large integers λ, n, every input $\boldsymbol{x} \in \mathsf{Im}(\mathcal{D}_n)$, every $(\mathsf{sk}, \mathsf{ek})$ in the image of $\mathsf{HSS.Gen}(1^\lambda)$, and every (s_0, s_1) in the image of $\mathsf{HSS.Share}(\mathsf{sk}, \boldsymbol{x})$, it holds that $|s_\sigma| \leq \mathsf{comp}(\lambda, n)$ for $\sigma = 0, 1$.*

由LPN和Degree-2 CHSS构造PCG：

准备：

Let $\mathcal{R}$ be a ring. Let $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}(\mathcal{R})_n\}_{n\in\mathbb{N}}$ be a family of efficiently sampleable distributions over $\mathcal{R}^n$. Let comp be a compression ratio. Let $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be a (2-party secret-key) degree-2 $(\mathcal{D}(\mathcal{R}) \times \mathcal{D}(\mathcal{R}), \mathsf{comp})$-CHSS. We describe below a construction of a bilinear correlation generator over $\mathcal{R}$. Note that any bilinear function $B : \mathcal{R}^n \times \mathcal{R}^n \mapsto \mathcal{R}^m$ can be fully described by a list of m matrices $B_1 \cdots B_m$ with $B_i \in \mathcal{R}^{n\times n}$ such that for any inputs $(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{R}^n \times \mathcal{R}^n$, $B(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\intercal B_i \boldsymbol{x}')_{i\leq m}$. Let α be a positive constant.

构造：

- $\mathsf{BCG.Setup}(1^\lambda)$: output $(\mathsf{pp} = \mathsf{ek}, \mathsf{sk}) \xleftarrow{\$} \mathsf{HSS.Gen}(1^\lambda)$.
- $\mathsf{BCG.Gen}(\mathsf{sk}, B)$: let n denote the input size of B and let $\ell \leftarrow \alpha n$. Let $M \leftarrow \mathbf{C}(\ell, \ell - n, \mathcal{R})$ be the encoding matrix of a linear code, and let $N \in \mathcal{R}^{n\times\ell}$ denote its parity check matrix (*i.e.*, N is the matrix over $\mathcal{R}^{n\times\ell}$ which satisfies $NM = 0$). Pick two vectors $(\boldsymbol{z}_0, \boldsymbol{z}_1) \leftarrow \mathcal{D}_\ell(\mathcal{R}) \times \mathcal{D}_\ell(\mathcal{R})$ and let r_0 (resp. r_1) denote the random coin used to sample $\boldsymbol{z}_0$ (resp. $\boldsymbol{z}_1$). Run $(s_0, s_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, (\boldsymbol{z}_0, \boldsymbol{z}_1))$. Output $\mathsf{k}_\sigma \leftarrow (r_\sigma, s_\sigma)$ for $\sigma = 0, 1$.
- $\mathsf{BCG.Expand}(\mathsf{pp}, \sigma, \mathsf{k}_\sigma)$: parse k_σ as (r_σ, s_σ) and reconstruct $\boldsymbol{z}_\sigma$ from r_σ (using the sampling procedure of $\mathcal{D}_n(\mathcal{R})$). Define the bilinear function $P : \mathcal{R}^\ell \times \mathcal{R}^\ell \mapsto \mathcal{R}^m$ as follows: on input $(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{R}^\ell \times \mathcal{R}^\ell$, P outputs $(\boldsymbol{x}^\intercal \cdot B'_i \cdot \boldsymbol{x}')_{i\leq m}$, where B'_i is defined as $B'_i \leftarrow N^\intercal B_i N$. Set $\boldsymbol{x}_\sigma \leftarrow N\boldsymbol{z}_\sigma$. Compute $\boldsymbol{y}_\sigma \leftarrow \mathsf{HSS.Eval}(\sigma, \mathsf{pp}, s_\sigma, P)$, and output $(\boldsymbol{x}_\sigma, \boldsymbol{y}_\sigma)$.

定理陈述和证明：

Theorem 47. *Assuming the $(\mathcal{D}(\mathcal{R}), \mathbf{C})$-$\mathsf{LPN}(\ell, \ell - n)$ assumption, the above construction is a bilinear correlation generator with seed size upper-bounded by $2 \cdot \mathsf{comp}(\lambda, \ell)$.*

C.5.1 Correctness.

As $(\boldsymbol{z}_0, \boldsymbol{z}_1)$ is sampled from $\mathcal{D}_\ell(\mathcal{R}) \times \mathcal{D}_\ell(\mathcal{R})$, the relaxed correctness property of the CHSS applies, and we get:

$$\begin{aligned}\boldsymbol{y}_0 + \boldsymbol{y}_1 &= \mathsf{HSS.Eval}(0, \mathsf{ek}, s_0, P) + \mathsf{HSS.Eval}(1, \mathsf{ek}, s_1, P) \\ &= (\boldsymbol{z}_0^\intercal B'_i \boldsymbol{z}_1)_{i \le m} = ((N\boldsymbol{z}_0)^\intercal B_i (N\boldsymbol{z}_1))_{i \le m} \\ &= (\boldsymbol{x}_0^\intercal B_i \boldsymbol{x}_1)_{i \le m} = B(\boldsymbol{x}_0, \boldsymbol{x}_1).\end{aligned}$$

C.5.2 Security.

Let $\mathcal{A}$ be a (stateful, nonuniform) PPT adversary, and let σ be a bit. We proceed through a sequence of game.

- **Game G_0.** In this game, we run $(\mathsf{pp}, \mathsf{sk}) \xleftarrow{\$} \mathsf{BCG.Setup}(1^\lambda)$, set $B \leftarrow \mathcal{A}(\mathsf{pp})$, $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{BCG.Gen}(\mathsf{sk}, B)$, and
$$(\boldsymbol{x}_\sigma, \boldsymbol{y}_\sigma) \leftarrow \mathsf{BCG.Expand}(\mathsf{pp}, \sigma, \mathsf{k}_\sigma).$$
This corresponds to the first experiment in the security definition of bilinear correlation generators. Let b_0 be the output of $\mathcal{A}(\boldsymbol{x}_\sigma, \mathsf{k}_{1-\sigma})$.
- **Game G_1.** In this game, we modify the execution of $\mathsf{BCG.Gen}$ as follows: instead of computing $(s_0, s_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, (\boldsymbol{z}_0, \boldsymbol{z}_1))$, we set $(s_0, s_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, 0^{2\ell})$. Let b_1 denote the output of $\mathcal{A}$ in this game. By the security property of the CHSS, the distribution of $(\mathsf{ek}, s_{1-\sigma})$ in this game is computationally indistinguishable from the distribution of $(\mathsf{ek}, s_{1-\sigma})$ in G_0, hence we have $\Pr[b_0 \neq b_1] = \mathsf{negl}(\lambda)$.
- **Game G_2.** In this game, we modify the execution of $\mathsf{BCG.Expand}$ as follows: instead of computing $\boldsymbol{x}_\sigma$ as $N\boldsymbol{z}_\sigma$, we pick $\boldsymbol{x}_\sigma \xleftarrow{\$} \mathcal{R}^n$. Note that $\mathsf{k}_{1-\sigma}$ does not depend on $\boldsymbol{z}_\sigma$, hence distinguishing between the games G_2 and G_1 amounts to distinguishing $N\boldsymbol{z}_\sigma$ from a random vector over $\mathcal{R}^n$, where $\boldsymbol{z}_\sigma$ is drawn from $\mathcal{D}_\ell(\mathcal{R})$. Note also that $N\boldsymbol{z}_\sigma = N(M\boldsymbol{a} + \boldsymbol{z}_\sigma)$ for any vector $\boldsymbol{a} \in \mathcal{R}^n$ (as $NM = 0$). Therefore, this amounts to distinguishing $M\boldsymbol{a} + \boldsymbol{z}_\sigma$ from random, for an arbitrary secret vector $\boldsymbol{a} \in \mathcal{R}^n$, and a noise vector $\boldsymbol{z}_\sigma$ sampled from $\mathcal{D}_\ell(\mathcal{R})$, which is infeasible under the $(\mathcal{D}(\mathcal{R}), \mathbf{C}) - \mathsf{LPN}(\ell, \ell - n)$ assumption. Therefore, denoting b_2 the output of $\mathcal{A}$ in G_2, we have $\Pr[b_1 \neq b_2] = \mathsf{negl}(\lambda)$.
- **Game G_3.** In this game, we revert the change made in G_1 and compute again (s_0, s_1) as $\mathsf{HSS.Share}(\mathsf{sk}, (\boldsymbol{z}_0, \boldsymbol{z}_1))$. Let b_3 denote the output of $\mathcal{A}$ in this game. By the security property of the CHSS, we have $\Pr[b_2 \neq b_3] = \mathsf{negl}(\lambda)$. Furthermore, this game is exactly the second experiment in the security definition of a BCG, which concludes the proof.

C.5.3 Efficiency.

The seed size of the BCG is $|\mathsf{k}_\sigma| = |r_\sigma| + |s_\sigma|$. By the compressibility of the CHSS, $|s_\sigma| \leq \mathsf{comp}(\lambda, \ell)$. Furthermore, by the restricted correctness of the CHSS, it must hold that $|r_\sigma| \leq |s_\sigma| \leq \mathsf{comp}(\lambda, \ell)$ (otherwise, using HSS.Share and HSS.Eval would allow to compress samples from $\mathcal{D}_\ell(\mathcal{R})$ below their amount of entropy, which is impossible). Hence, we get $|\mathsf{k}_\sigma| \leq 2 \cdot \mathsf{comp}(\lambda, \ell)$.

由LPN和Compressible Las Vegas HSS构造Sanitizable BCG：

方法和上面一样，只需注意一个小区别：按照定义，Las Vegas HSS是对于整个输出有一个整体的失败概率，而sanitizable BCG对于每个输出都有一个flag（因为我们要挑出错误的输入然后改正），事实上这没有本质区别：

> This is essentially a syntactic difference: existing construction of Las Vegas HSS can easily be modified to output a flag for each output bit. In the formal construction of sanitizable BCG from Las Vegas HSS, it suffices to apply the Las Vegas HSS independently for each functions f_i outputting the ith bit of the target bilinear correlation, to get independent failure flags for each output bit.

Group-Based HSS for Bilinear Functions:

Below, we briefly recall the HSS scheme of [BGI16a], taking into account the optimizations for bilinear functions of [BCG+17].

记号：

Notations. For vectors $\boldsymbol{x}_i$ over a multiplicative group, we denote by $\prod_i \boldsymbol{x}_i$ their component-wise product. Given a multiplicative group $\mathbb{G}$ of order q, a length-n vector $\boldsymbol{g} = (g_1, \cdots, g_n) \in \mathbb{G}^n$, and a length-$n$ vector of exponents $\boldsymbol{x} = (x_1, \cdots, x_n) \in \mathbb{Z}_q^n$, we denote $\boldsymbol{x} \bullet \boldsymbol{g}$ the "scalar product in the exponent": $\boldsymbol{x} \bullet \boldsymbol{g} = \prod_{i=1}^n g_i^{x_i}$; for any $g \in \mathbb{G}$, $g^{\boldsymbol{x}}$ denotes $(g^{x_1}, \cdots, g^{x_n})$.

首先，对于有限群的元素，有几种encoding：

Let q be a large prime, and let $\mathbb{G}$ be a hard-discrete-log group of order q. Let g denote a generator of $\mathbb{G}$. For any $x \in \mathbb{Z}_q$, we consider the following 3 types of two-party encodings:

LEVEL 1: "Encryption." For $x \in \mathbb{Z}_q$, we let $[x]$ denote g^x, and $[\![x]\!]_s$ denote $([r], [r \cdot s + x])$ for a uniformly random $r \in \mathbb{Z}_q$, which corresponds to an ElGamal encryption of x with a secret key $s \in \mathbb{Z}_q$. All level-1 encodings are known to both parties. We let $\mathsf{sk} \leftarrow (s, -1)$.

LEVEL 2: "Additive shares." Let $\langle x \rangle$ denote a pair of shares $x_0, x_1 \in \mathbb{Z}_q$ such that $x_0 = x_1 + x$, where each share is held by a different party. We let $\langle\!\langle x \rangle\!\rangle_s$ denote $(\langle -s \cdot x \rangle, \langle x \rangle) = \mathsf{sk} \cdot \langle x \rangle \in (\mathbb{Z}_q^2)^2$, namely each party holds one share of $\langle -s \cdot x \rangle$ and one share of $\langle x \rangle$. Note that both types of encodings are additively homomorphic over $\mathbb{Z}_q$, namely given encodings of x and x' the parties can locally compute a valid encoding of $x + x'$.

LEVEL 3: "Multiplicative shares." Let $\{x\}$ denote a pair of shares $x_0, x_1 \in \mathbb{G}$ such that the difference between their discrete logarithms is x. That is, $x_0 = x_1 \cdot g^x$.

现在，显然可以locally计算线性函数，为了计算degree-2 polynomials，我们希望计算乘法，方法如下（类似pairing-based encryption）：

of xy. To this end, we consider the following two types of operations, performed locally by the two parties:

1. $\mathsf{Pair}([\![x]\!]_s, \langle\!\langle y \rangle\!\rangle_s) \mapsto \{xy\}$. This pairing operation exploits the fact that the decryption of an ElGamal ciphertext $[\![x]\!]_s$ is computed as a scalar product in the exponent: $g^x = \mathsf{sk} \bullet [\![x]\!]_s$. Therefore, Pair computes $\langle\!\langle y \rangle\!\rangle_s \bullet [\![x]\!]_s = \langle y \rangle \bullet (\mathsf{sk} \bullet [\![x]\!]_s) = \{xy\}$. Note that we consider ElGamal ciphertexts for the sake of concreteness only; any encryption scheme whose decryption follows a similar "scalar product in the exponent" structure would suffice.
2. $\mathsf{Convert}(\{z\}, \delta) \mapsto \langle z \rangle$, with failure bound δ. The implementation of Convert is also given an upper bound M on the "payload" z ($M = 1$ by default), and its expected running time grows linearly with M/δ. We omit M from the following notation.

（Convert算法在P54上部）

把Pair和Convert结合起来就得到Mult乘法，但是输出不是level1或level2的encoding，所以只能做双线性，不能更高度数函数：

Given the Pair and Convert algorithms, the multiplication algorithm Mult sequentially executes these two operations: $\mathsf{Mult}([\![x]\!]_c, \langle\!\langle y \rangle\!\rangle_c, \delta) \mapsto \langle xy \rangle$, with error δ. Note that the output of the procedure is not a level 1 or a level 2 encoding. Therefore, this δ-failure HSS allows to evaluate arbitrary bilinear functions B on vectors $(\boldsymbol{x}, \boldsymbol{y})$ (encodings of level 1 and 2, as well as additive shares, being additively homomorphic) but does not generalize immediately to functions of higher degree; generalizations to branching programs are presented in [BGI16a,BGI17,BCG+17], but are several orders of magnitude less efficient.

The BGN-EG Cryptosystem

BGN 是一种基于椭圆曲线群的pairing-based加密，可以同态计算二次多项式。
我们用的是Freeman的版本[Fre10]的简化版。

> The Boneh-Goh-Nissim cryptosystem (BGN) was introduced in [BGN05]. It is a variant of the ElGamal cryptosystem over composite-order pairing-friendly elliptic curve, which allows to homomorphically compute any degree-2 polynomial on encrypted plaintexts, provided that the output is of polynomial size (as decryption requires computing a discrete logarithm).

具体加密算法：

准备过程：

Let $\mathsf{BilinearGen}$ denote a PPT algorithm which, on input 1^λ, outputs a prime q, the description of three cyclic groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t)$, elements $(g_1, g_2) \in \mathbb{G}_1 \times \mathbb{G}_2$, and a map e such that

- the cyclic groups have the same order $q = q(\lambda)$;
- the map $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_t$ is an efficiently computable non-degenerate bilinear map, *i.e.*, $\forall (u, v) \in \mathbb{G}_1 \times \mathbb{G}_2$ and $(a, b) \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$;
- g_i generates $\mathbb{G}_i$ for $i \in \{1, 2\}$ (and $g_t \leftarrow e(g_1, g_2)$ generates $\mathbb{G}_t$).

Note that this captures groups equipped with an *asymmetric* pairing. To simplify notations, given vectors $\boldsymbol{x} \in \mathbb{G}_1^n, \boldsymbol{y} \in \mathbb{G}_2^n$, we will write $e(\boldsymbol{x}, \boldsymbol{y})$ to denote $(e(x_i, y_j))_{i,j \le n}$. When it happens that the components of the vector are vectors themselves (e.g. if $\boldsymbol{x}, \boldsymbol{y}$ are vectors of ciphertexts, each ciphertext consisting of several group elements), we apply the notation recursively: $e(\boldsymbol{x}, \boldsymbol{y}) = (e(x_i, y_j))_{i,j \le n}$ and for every i, j, $e(x_i, y_j) = (e(x_{i,i'}, y_{j,j'}))_{i',j'}$.

加密步骤：

We outline below our simplified variant of Freeman's adaptation of the BGN cryptosystem, which we denote BGN-EG.

- $\mathsf{BGN\text{-}EG.Setup}(1^\lambda)$: output $\mathsf{pp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, e) \xleftarrow{\$} \mathsf{BilinearGen}(1^\lambda)$.
- $\mathsf{BGN\text{-}EG.KeyGen}(\mathsf{pp})$: pick $(s_1, s_2) \xleftarrow{\$} \mathbb{Z}_q^2$, compute $(h_1, h_2) \leftarrow (g_1^{s_1}, g_2^{s_2})$. Set $\mathsf{sk}_i \leftarrow (s_i, -1)$ for $i = 1, 2$, and $\mathsf{sk}_t = (s_1 \cdot s_2, -s_1, -s_2, 1)$. Output $\mathsf{pk} \leftarrow (\mathsf{pp}, h_1, h_2)$ and $\mathsf{sk} \leftarrow (\mathsf{sk}_1, \mathsf{sk}_2, \mathsf{sk}_t)$.
- $\mathsf{BGN\text{-}EG.Enc}_i(\mathsf{pk}, m; r)$: on input the public key pk, a message $m \in \mathbb{Z}_q$, and a random coin $r \in \mathbb{Z}_q$, output $\boldsymbol{c}_i \leftarrow (g_i^r, h_i^r g_i^m)$ (this corresponds to encryption over $\mathbb{G}_i$).
- $\mathsf{BGN\text{-}EG.Dec}_i(\mathsf{sk}, \boldsymbol{c}_i)$: on input the secret key sk and a ciphertext $\boldsymbol{c}_i$, compute $c \leftarrow \mathsf{sk}_i \bullet \boldsymbol{c}_i$ and output $m \leftarrow \mathsf{dlog}_{g_i}(c)$.

正确性、安全性、线性同态、如何实现乘法同态：

Correctness follows by inspection; security reduces to the decisional Diffie-Hellman assumption (DDH) in $\mathbb{G}_1$ and $\mathbb{G}_2$. It is easy to see that the scheme is additively homomorphic over each of $\mathbb{G}_1$ and $\mathbb{G}_2$. Furthermore, given an encryption $\boldsymbol{c}_1$ of a message m_1 over $\mathbb{G}_1$ and an encryption $\boldsymbol{c}_2$ of a message m_2 over $\mathbb{G}_2$, one can homomorphically construct an encryption of $m_1 m_2$ over $\mathbb{G}_t$ as follows: compute $\boldsymbol{c}_t \leftarrow e(\boldsymbol{c}_1, \boldsymbol{c}_2)$. The resulting ciphertext has four components and remains additively homomorphic over $\mathbb{G}_t$ (addition of plaintext is computed by component-wise multiplication). Decryption of a ciphertext over $\mathbb{G}_t$ is performed by computing $c \leftarrow \mathsf{sk}_t \bullet \boldsymbol{c}_t$, and outputting $m \leftarrow \mathsf{dlog}_{g_t}(c)$, where $g_t = e(g_1, g_2)$.

Compressible HSS from BGN-EG：

输入比较稀疏，（with sparse structure），然后level1和level2的encoding可以压缩。

We now show how the group-based HSS of [BGI16a, BGI17, BCG+17] can be modified to get a compressible HSS. In this section, we will consider compressible HSS over a small (polynomial-size) ring $\mathbb{Z}_t$ for inputs drawn from the Bernouilli distribution $\mathsf{Ber}_r(\mathbb{Z}_t)$ for some rate r (see Section 3.3); that is, we will consider inputs with a sparse structure, and show how level 1 and level 2 encodings of such inputs can be compressed. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t)$ denote three cyclic groups of order q, and let $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_t$ denote a pairing. Observe that the ciphertexts in the target group $\mathbb{G}_t$ of BGN-EG have a suitable structure to be used as level-1 encodings in the HSS scheme, as decryption of a ciphertext over $\mathbb{G}_t$ is performed via a scalar product in the exponent.

改进上面的BGN-EN，实现乘法的additive share：

We modify the HSS scheme of the previous section as follows: a level 1 encoding $[\![x]\!]_{\mathsf{sk}}$ of x is a BGN-EG encryption of x in the target group $\mathbb{G}_t$. A level 2 encoding $\langle\!\langle y \rangle\!\rangle_{\mathsf{sk}}$ of a message y is a 4-tuple $\mathsf{sk}_t \cdot \langle y \rangle$. The Pair algorithm, on input a level 1 encoding $[\![x]\!]_{\mathsf{sk}}$ and level 2 shares $\langle\!\langle y \rangle\!\rangle_{\mathsf{sk}}$, outputs $\langle\!\langle y \rangle\!\rangle_{\mathsf{sk}} \bullet [\![x]\!]_{\mathsf{sk}}$; a level 3 encoding of a message z is a multiplicative share over $\mathbb{G}_t$ of g_t^z; it follows that $\mathsf{Pair}([\![x]\!]_{\mathsf{sk}}, \langle\!\langle y \rangle\!\rangle_{\mathsf{sk}}) = \{xy\}$. Then, the parties can apply the optimized Convert procedure of [DKK18] to obtain additive shares of xy. The security of this modified HSS scheme immediatly reduces to the DDH assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$, by the same security analysis as for the HSS of [BGI16a] (note that, because we restrict our attention to HSS for

对level1的压缩：

C.8.2 Compressing Level 1 Encodings under BGN-EG.

The modified scheme suggests a natural strategy to reduce the size of level 1 encodings when the input is sparse, by exploiting the homomorphic properties of BGN-EG ciphertexts. Let $\boldsymbol{m}$ be a length-ℓ vector over $\mathbb{Z}_q$, which has at most k non-zero coordinates. We assume ℓ to be a square for simplicity. The compression method works as follows: arrange the coordinates of $\boldsymbol{m}$ in a $\sqrt{\ell} \times \sqrt{\ell}$ matrix M. Decompose M into $\sum_{i=1}^{k} M_i$, where each matrix M_i has a single non-zero coordinate. For $i = 1$ to k, let $(u_i, v_i) \in [\sqrt{\ell}] \times [\sqrt{\ell}]$ denote the coordinate of the non-zero entry of M_i.

Pick $2\sqrt{\ell}$ elements $(\alpha_{i,j}, \beta_{i,j})_{j \le \sqrt{\ell}}$ of $\mathbb{Z}_q$ as follows: for each pair $(u, v) \neq (u_i, v_i)$, set $\alpha_{i,u} = \beta_{i,v} = 0$, and set $\alpha_{i,u_i} = 1$, $\beta_{i,v_i} = M_i|_{u_i,v_i}$. Observe that, by construction, it holds that $M_i|_{u,v} = \alpha_{i,u} \cdot \beta_{i,v}$ for any pair $(u, v) \in [\sqrt{\ell}] \times [\sqrt{\ell}]$. Therefore, we have for any $(u, v) \in [\sqrt{\ell}] \times [\sqrt{\ell}]$:

$$M|_{u,v} = \sum_{i=1}^{k} \alpha_{i,u} \cdot \beta_{i,v}.$$

This shows that for any vector $\boldsymbol{m}$ with at most k non-zero entries, one can create a level 1 encoding of $\boldsymbol{m}$ as follows: compute the $(\alpha_{i,u}, \beta_{i,u})_{i \le k, u \le \sqrt{\ell}}$ as above, and set the encoding of $\boldsymbol{m}$ to be $k \cdot \sqrt{\ell}$ BGN-EG encryptions of $(\alpha_{i,u})_{i,u}$ over $\mathbb{G}_1$, and $k \cdot \sqrt{\ell}$ BGN-EG encryptions of $(\beta_{i,u})_{i,u}$ over $\mathbb{G}_2$. Using the homomorphic properties of BGN-EG, any party can then locally reconstruct a BGN-EG encryption of $\boldsymbol{m} = (\sum_{i=1}^{k} \alpha_{i,u} \cdot \beta_{i,v})_{u,v}$ over $\mathbb{G}_t$. The size of the compressed encoding is $2k \cdot \sqrt{\ell} \cdot (|\mathbb{G}_1| + |\mathbb{G}_2|)$. Note that this strategy corresponds exactly to using the LPN-based PRG introduced in Section 4.4 to stretch the encoded seed.

对level2的压缩：

C.8.3 Compressing Level 2 Encodings using FSS.

We now turn our attention to level 2 encodings $\langle\!\langle \boldsymbol{m} \rangle\!\rangle_{\mathsf{sk}} = (\langle s_1 s_2 \cdot \boldsymbol{m} \rangle, \langle -s_1 \cdot \boldsymbol{m} \rangle, \langle -s_2 \cdot \boldsymbol{m} \rangle, \langle \boldsymbol{m} \rangle)$. Note that if $\boldsymbol{m}$ is a sparse vector, so are $s_1 s_2 \cdot \boldsymbol{m}$, $-s_1 \cdot \boldsymbol{m}$, and $-s_2 \cdot \boldsymbol{m}$. We therefore focus on compressing additive shares of arbitrary sparse vectors over $\mathbb{Z}_q$. As was observed recently in [BCGI18], this type of correlation can be efficiently compressed using function secret sharing for multi-point functions (MPFSS). We elaborate below.

Let $\mathsf{MPFSS} = (\mathsf{MPFSS.Gen}, \mathsf{MPFSS.Eval}, \mathsf{MPFSS.FullEval})$ be a multi-point function secret sharing. On input a vector $\boldsymbol{m} \in \mathbb{Z}_q^\ell$ with $\mathsf{HW}(\boldsymbol{m}) \leq k$, let $\mathsf{proj}_{\boldsymbol{m}} : [\ell] \mapsto \mathbb{Z}_q$ be the function which, on input $i \in [\ell]$, outputs the i'th coordinate of $\boldsymbol{m}$. Note that $\mathsf{proj}_{\boldsymbol{m}}$ is an (ℓ, k)-multi-point function over $\mathbb{Z}_q$. To generate compressed additive shares of $\boldsymbol{m}$, compute $(K_0, K_1) \xleftarrow{\$} \mathsf{MPFSS.Gen}(1^\lambda, \mathsf{proj}_{\boldsymbol{m}})$. To decompress the string, each party with input K_σ computes $\mathsf{MPFSS.FullEval}(\sigma, K_\sigma)$, obtaining additive shares of $(\mathsf{proj}_{\boldsymbol{m}}(i))_{i \in [\ell]} = \boldsymbol{m}$. Correctness immediately follows from the correctness of the underlying MPFSS. Regarding security, we must show that for any pair $(\boldsymbol{m}, \boldsymbol{m'})$ (with $\mathsf{HW}(\boldsymbol{m}), \mathsf{HW}(\boldsymbol{m'}) \leq k$) and any $\sigma \in \{0, 1\}$, the distribution of $(\mathsf{ek}_\sigma, s_\sigma)$ obtained by sampling $(\mathsf{sk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{HSS.Gen}(1^\lambda)$, sampling $(s_0, s_1) \leftarrow \mathsf{HSS.Enc}(\mathsf{sk}, \boldsymbol{m})$ or $(s_0, s_1) \leftarrow \mathsf{HSS.Enc}(\mathsf{sk}, \boldsymbol{m'})$, and outputting $(\mathsf{ek}_\sigma, s_\sigma)$, are computationally indistinguishable. This immediately follows from the fact that, by the MPFSS security, there is a simulator which (given $\mathsf{Leak}(\mathsf{proj}_{\boldsymbol{m}}) = \mathsf{Leak}(\mathsf{proj}_{\boldsymbol{m'}})$, and no further information about $\boldsymbol{m}, \boldsymbol{m'}$) can output a simulated key $s_\sigma = K_\sigma$ whose distribution is indistinguishable from an honestly generated key.

Using the PRG-based MPFSS of [BGI16b, BCGI18], the size of a compressed encoding using this method is equal to $k \cdot (\lceil \log \ell \rceil \cdot (\lambda + 2) + \lambda + \lceil \log q \rceil)$. We refer the reader to [BCGI18] for further discussions on this method and optimizations of MPFSS tailored to this application.

结合以上结果：

得到DDH-based CHSS：

Theorem 48. *Assuming the DDH assumption over pairing-friendly elliptic curves, for any integer t of polynomial size and any integer k, there exists a $(\mathsf{Ber}_{k/\ell}(\mathbb{Z}_t), \mathsf{comp}(\lambda,\ell,k))$-compressible (Las Vegas, secret-key, degree-2) CHSS, with*

$$\mathsf{comp}(\lambda,\ell,k) = k\cdot\sqrt{\ell}\cdot\mathsf{poly}(\lambda).$$

For self-containement, we describe below the full CHSS. It has two sharing algorithms, corresponding to level 1 and level 2 shares respectively. The evaluation procedure allows to compute (shares of) any bilinear function $B(\boldsymbol{x},\boldsymbol{y})$ where $\boldsymbol{x}$ is level-1-shared and $\boldsymbol{y}$ is level-2-shares between the parties.

- $\mathsf{HSS.Gen}(1^\lambda)$: run $\mathsf{pp} = (q,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t,g_1,g_2,e) \xleftarrow{\$} \mathsf{BGN\text{-}EG.Setup}(1^\lambda)$, as well as $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{BGN\text{-}EG.KeyGen}(\mathsf{pp})$. Compute $g \leftarrow e(g_1,g_2)$. Output $\mathsf{ek} \leftarrow (\mathsf{pp},\mathsf{pk},g)$ and sk.
- $\mathsf{HSS.Share}_1(\mathsf{sk},\boldsymbol{m})$: let k denote an upper bound on the sparsity of $\boldsymbol{m}$, and let ℓ denote the length of $\boldsymbol{m}$. Let $(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)_{i\le k}$ denote the decomposition of $\boldsymbol{m}$ in $2k$ length-$\sqrt{\ell}$ vectors, as described in Section C.8. Compute, for $i=1$ to k, $(\boldsymbol{c}_i,\boldsymbol{d}_i) \xleftarrow{\$} (\mathsf{BGN\text{-}EG.Enc}_1(\mathsf{pk},\boldsymbol{\alpha}_i), \mathsf{BGN\text{-}EG.Enc}_2(\mathsf{pk},\boldsymbol{\beta}_i))$ and output $\mathsf{share}_0 = \mathsf{share}_1 = (\boldsymbol{c}_i,\boldsymbol{d}_i)_{i\le k}$.
- $\mathsf{HSS.Share}_2(\mathsf{sk},\boldsymbol{m})$: let k denote an upper bound on the sparsity of $\boldsymbol{m}$, and let ℓ denote the length of $\boldsymbol{m}$. Parse sk as $(\mathsf{sk}_1,\mathsf{sk}_2,\mathsf{sk}_t)$ and parse sk_i as $(s_i,-1)$ for $i=1,2$. Let $\mathbf{proj}_{\boldsymbol{m},\mathsf{sk}} \leftarrow (\mathsf{proj}_{s_1s_2\boldsymbol{m}},\mathsf{proj}_{-s_1\boldsymbol{m}},\mathsf{proj}_{-s_2\boldsymbol{m}},\mathsf{proj}_{\boldsymbol{m}})$, where proj is defined as in Section C.8.3. Compute

  $$(\boldsymbol{K}_0,\boldsymbol{K}_1) \leftarrow \mathsf{MPFSS.Gen}(1^\lambda,\mathbf{proj}_{\boldsymbol{m},\mathsf{sk}}).$$

  Return $\mathsf{share}'_0 \leftarrow \boldsymbol{K}_0$ and $\mathsf{share}'_1 \leftarrow \boldsymbol{K}_1$.
- $\mathsf{HSS.Eval}(\sigma,\mathsf{ek},\mathsf{share}_\sigma,\mathsf{share}'_\sigma,B,\delta)$: On input party index $\sigma\in\{0,1\}$, evaluation key ek, level 1 share share_σ of a size-ℓ input, level 2 share share'_σ of a size-ℓ input, a bilinear function $B:\mathbb{Z}_t^\ell\times\mathbb{Z}_t^\ell\mapsto\mathbb{Z}_t^m$ with $B(\boldsymbol{x},\boldsymbol{y}) = (\sum_{i,j} b_{i,j,\theta}\cdot x_iy_j)_{\theta\le m}$, and failure probability bound $\delta>0$:
  - Parse share_σ as $(\boldsymbol{c}_i,\boldsymbol{d}_i)_{i\le k}$, and compute ℓ ciphertexts $(e_1,\cdots,e_\ell) \leftarrow \prod_{i=1}^k e(\boldsymbol{c}_i,\boldsymbol{d}_i)$.
  - Parse share'_σ as $\boldsymbol{K}_\sigma$. Compute $\boldsymbol{K}'_\sigma \leftarrow \mathsf{MPFSS.FullEval}(\sigma,\boldsymbol{K}_\sigma)$. Note that $\boldsymbol{K}'_\sigma$ is a vector of ℓ length-4 vectors $\boldsymbol{K}'_{\sigma,i}$.
  - For every $(i,j)\in[\ell]^2$, $r_{i,j} \leftarrow \mathsf{Pair}(e_i,\boldsymbol{K}'_{\sigma,j}) = \boldsymbol{K}'_{\sigma,j}\bullet e_i$.
  - For $\theta=1$ to m, let $h_\theta \leftarrow (b_{i,j,\theta})_{i,j}\bullet(r_{i,j})_{i,j}$.
  - Output $(\mathsf{Convert}(h_i,\delta/m))_{i\le m}$, where $\mathsf{Convert}$ is run in base g over $\mathbb{G}_t$.

再代入到上面的LPN-based BCG，得到：

Theorem 49. *Assuming the DDH assumption over pairing-friendly elliptic curves and the $\mathsf{LPN}((\alpha-1)\cdot n,\alpha\cdot n,k/(\alpha n))$ assumption over an integer ring $\mathbb{Z}_t$ of polynomial size, for a positive constant $\alpha>1$, there exists a δ-failure SBCG with seed size $k\cdot(\alpha\cdot n)^{1/2}\cdot\mathsf{poly}(\lambda)$.*

PCG from Lattices

这一部分扩展到任意次数有界多项式的correlation。

作为一个具体情况，我们考虑authenticated Beaver triples。

> In this section we give a lattice-based PCG construction for any family of polynomials of bounded degree over large finite fields, extending the results of the previous sections to more general correlations. As a use-case we consider the generation of authenticated Beaver triples

何为authenticated Beaver triple？：

[BCG+19b]：

$$\{(a, b, ab, a\alpha, b\alpha, ab\alpha) \mid a, b \in \mathbb{Z}_p\}$$

for some fixed MAC $\alpha \in \mathbb{Z}$

[BCGI20]:

Secret-sharing with MACs. We use authenticated secret-sharing based on SPDZ MACs between n parties, where a secret-sharing of $x \in \mathbb{Z}_p$ is defined as:

$$[\![x]\!] = (\alpha_i, x_i, m_{x,i})_{i=1}^n \quad \text{such that } \sum_i x_i = x, \sum_i m_{x,i} = x \cdot \sum_i \alpha_i$$

Note that the MAC key shares α_i are fixed for every shared x. The MAC shares $m_{x,i}$ are used to prevent a sharing from being opened incorrectly, via a MAC check procedure from [DKL+13]. An *authenticated multiplication triple* is a tuple of random sharings $([\![x]\!], [\![y]\!], [\![z]\!])$, where $x, y \xleftarrow{\$} \mathbb{Z}_p$ and $z = x \cdot y$. Our PCG outputs a single multiplication triple over the ring R_p, for $n = 2$ parties, together with additive shares of the MAC key $\alpha \in \mathbb{Z}_p$. When using the fully-reducible variant of ring-LPN, this is equivalent to N triples over $\mathbb{F}_{p^d}$ (where for suitably chosen p we can have $d = 1$).

（问题：所以这两个文章到底讨论的是不是一个correlation？这个alpha要不要share？）

基于的PRG和HSS：

> The assumptions we build on are the sparse **MQ-assumption** discussed in Section 7.1 and the ring-version of the learning with errors assumption

Definition 53 (Learning With Errors over Rings (RLWE)). *Let $N \in \mathbb{N}$ be a power of two $q \in \mathbb{N}$ with $q \geq 2$, $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_q = \mathcal{R}/(q\mathcal{R})$. Let χ be an error distribution over $\mathcal{R}$. Let $s \leftarrow \chi$. The $\mathsf{RLWE}_{N,q,\chi}$-assumption states that the following two distributions over $\mathcal{R}_q^2$ are computationally indistinguishable:*

- *$\mathcal{O}_{\chi,s}$: Output (a, b) where $a \leftarrow \mathcal{R}_q, e \leftarrow \chi$ and $b = a \cdot s + e$*
- *U: Output $(a, u) \leftarrow \mathcal{R}_q^2$*

关于**BGV2**（(Leveled) fully homomorphic encryption without bootstrapping.）：

Abstract

We present a novel approach to fully homomorphic encryption (FHE) that dramatically improves performance and bases security on weaker assumptions. A central conceptual contribution in our work is a new way of constructing leveled fully homomorphic encryption schemes (capable of evaluating arbitrary polynomial-size circuits), *without Gentry's bootstrapping procedure.*

Specifically, we offer a choice of FHE schemes based on the learning with error (LWE) or Ring LWE (RLWE) problems that have 2^λ security against known attacks. We construct:

- A leveled FHE scheme that can evaluate depth-L arithmetic circuits (composed of fan-in 2 gates) using $\tilde{O}(\lambda \cdot L^3)$ per-gate computation. That is, the computation is *quasi-linear* in the security parameter. Security is based on RLWE for an approximation factor exponential in L. This construction does not use the bootstrapping procedure.
- A leveled FHE scheme that can evaluate depth-L arithmetic circuits (composed of fan-in 2 gates) using $\tilde{O}(\lambda^2)$ per-gate computation, which is *independent of* L. Security is based on RLWE for *quasi-polynomial* factors. This construction uses bootstrapping *as an optimization.*

We obtain similar results for LWE, but with worse performance. All previous (leveled) FHE schemes required a per-gate computation of $\tilde{\Omega}(\lambda^{3.5})$, and all of them relied on sub-exponential hardness assumptions.

其中LWE和RLWE的介绍：

The learning with errors (LWE) problem was introduced by Regev [Reg05]. It is defined as follows.

Definition 5 (LWE)**.** *For security parameter λ, let $n = n(\lambda)$ be an integer dimension, let $q = q(\lambda) \geq 2$ be an integer, and let $\chi = \chi(\lambda)$ be a distribution over $\mathbb{Z}$. The $\mathsf{LWE}_{n,q,\chi}$ problem is to distinguish the following two distributions: In the first distribution, one samples $(\mathbf{a}_i, b_i)$ uniformly from $\mathbb{Z}_q^{n+1}$. In the second distribution, one first draws $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ uniformly and then samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^{n+1}$ by sampling $\mathbf{a}_i \leftarrow \mathbb{Z}_q^n$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = \langle \mathbf{a}, \mathbf{s} \rangle + e_i$. The $\mathsf{LWE}_{n,q,\chi}$ assumption is that the $\mathsf{LWE}_{n,q,\chi}$ problem is infeasible.*

Regev [Reg05] proved that for certain moduli q and Gaussian error distributions χ, the $\mathsf{LWE}_{n,q,\chi}$ assumption is true as long as certain worst-case lattice problems are hard to solve using a quantum algorithm. We state this result using the terminology of B-bounded distributions, which is a distribution over the integers where the magnitude of a sample is bounded with high probability. A definition follows.

Definition 6 (B-bounded distributions)**.** *A distribution ensemble $\{\chi_n\}_{n \in \mathbb{N}}$, supported over the integers, is called B-bounded if*

$$\Pr_{e \leftarrow \chi_n}[|e| > B] = \mathsf{negl}(n) \ .$$

We can now state Regev's worst-case to average-case reduction for LWE.

Theorem 2 (Regev [Reg05])**.** *For any integer dimension n, prime integer $q = q(n)$, and $B = B(n) \geq 2n$, there is an efficiently samplable B-bounded distribution χ such that if there exists an efficient (possibly quantum) algorithm that solves $\mathsf{LWE}_{n,q,\chi}$, then there is an efficient quantum algorithm for solving $\tilde{O}(qn^{1.5}/B)$-approximate worst-case SIVP and gapSVP.*

Peikert [Pei09] de-quantized Regev's results to some extent – that is, he showed the $\mathsf{LWE}_{n,q,\chi}$ assumption is true as long as certain worst-case lattice problems are hard to solve using a *classical* algorithm. (See [Pei09] for a precise statement of these results.)

Applebaum et al. [ACPS09] showed that if LWE is hard for the above distribution of $\mathbf{s}$, then it is also hard when $\mathbf{s}$'s coefficients are sampled according to the noise distribution χ.

2.4 The Ring Learning with Errors (RLWE) Problem

The ring learning with errors (RLWE) problem was introduced by Lyubaskevsky, Peikert and Regev [LPR10]. We will use an simplified special-case version of the problem that is easier to work with [Reg10, BV11a].

Definition 7 (RLWE)**.** *For security parameter λ, let $f(x) = x^d + 1$ where $d = d(\lambda)$ is a power of 2. Let $q = q(\lambda) \geq 2$ be an integer. Let $R = \mathbb{Z}[x]/(f(x))$ and let $R_q = R/qR$. Let $\chi = \chi(\lambda)$ be a distribution over R. The $\mathsf{RLWE}_{d,q,\chi}$ problem is to distinguish the following two distributions: In the first distribution, one samples (a_i, b_i) uniformly from R_q^2. In the second distribution, one first draws $s \leftarrow R_q$ uniformly and then samples $(a_i, b_i) \in R_q^2$ by sampling $a_i \leftarrow \mathbb{R}_q$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = a_i \cdot s + e_i$. The $\mathsf{RLWE}_{d,q,\chi}$ assumption is that the $\mathsf{RLWE}_{d,q,\chi}$ problem is infeasible.*

The RLWE problem is useful, because the well-established shortest vector problem (SVP) over ideal lattices can be reduced to it, specifically:

Theorem 3 (Lyubashevsky-Peikert-Regev [LPR10])**.** *For any d that is a power of 2, ring $R = \mathbb{Z}[x]/(x^d + 1)$, prime integer $q = q(d) = 1 \bmod d$, and $B = \omega(\sqrt{d \log d})$, there is an efficiently samplable distribution χ that outputs elements of R of length at most B with overwhelming probability, such that if there exists an efficient algorithm that solves $\mathsf{RLWE}_{d,q,\chi}$, then there is an efficient quantum algorithm for solving $d^{\omega(1)} \cdot (q/B)$-approximate worst-case SVP for ideal lattices over R.*

Typically, to use RLWE with a cryptosystem, one chooses the noise distribution χ according to a Gaussian distribution, where vectors sampled according to this distribution have length only $\mathrm{poly}(d)$ with overwhelming probability. This Gaussian distribution may need to be "ellipsoidal" for certain reductions to go through [LPR10]. It has been shown for RLWE that one can equivalently assume that s is alternatively sampled from the noise distribution χ [LPR10].

PCG from Somewhat Homomorphic Encryption

想法是：首先从一个支持分布式解密的somewhat homomorphic encryption可以构造出HSS，然后套用之前的PRG+HSS方法论：

> As observed in [DHRW16, BKS19], from a somewhat homomorphic encryption scheme which supports distributed decryption one can construct a homomorphic secret sharing scheme.

定义Depth-d Somewhat Homomorphic Encryption with Distributed Decryption：

Definition 54 (Depth-d Somewhat Homomorphic Encryption w/ Distributed Decryption). *Let* $\mathsf{PKE} := (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an IND-CPA secure public-key encryption scheme. We say that* PKE *is a* secure depth-d public-key encryption scheme with distributed decryption *if it further satisfies the following properties:*

- Distributed decryption: *Let* $\mathcal{R} := \mathbb{Z}[X]/(X^N+1)$*, for* N *a power of two,* $\kappa \in \mathbb{N}$ *and the secret key space of* PKE *contained in* $\mathcal{R}_q^\kappa$*. We say* PKE *supports distributed decryption, if there exists an algorithm* DDec *such that for* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$*,* $\mathsf{sk}_0 \xleftarrow{\$} \mathcal{R}_q^\kappa$*,* $\mathsf{sk}_1 := \mathsf{sk} - \mathsf{sk}_0$*,* $m \in \mathcal{R}_p$*, and* $\boldsymbol{c} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m)$ *it holds* $\mathsf{DDec}(\mathsf{sk}_0, \boldsymbol{c}) + \mathsf{DDec}(\mathsf{sk}_1, \boldsymbol{c}) = m$ *with overwhelming probability.*
- Depth-d somewhat homomorphic encryption: *There exists a procedure* $\mathsf{PKE.Eval}$ *such that for any function* $f\colon \mathcal{R}^n \to \mathcal{R}^m$ *that can be evaluated by a circuit of depth at most* d*, for any* $\lambda \in \mathbb{N}$*, for any* $(\mathsf{pk}, \mathsf{sk})$ *in the image of* $\mathsf{Gen}(1^\lambda)$*, for all messages* $m_1, \ldots, m_n \in \mathcal{R}_p$*, for all ciphertexts* $\boldsymbol{c}_1, \ldots \boldsymbol{c}_n$ *in the image of* $\mathsf{PKE.Enc}(\mathsf{pk}, m_1), \ldots, \mathsf{PKE.Enc}(\mathsf{pk}, m_n)$ *and for any* $\boldsymbol{c}$ *in the image of* $\mathsf{PKE.Eval}(f, (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n))$ *it holds*

$$\mathsf{PKE.Dec}(\mathsf{sk}, \boldsymbol{c}) = f(m_1, \ldots, m_n).$$

  Notation. *We generalize* $\mathsf{Alg} \in \{\mathsf{Enc}, \mathsf{DDec}, \mathsf{Eval}\}$ *to vectors of inputs in a straightforward way:* Alg *is run independently on each entry of the vector (with independent random coins if* Alg *is randomized).*

套用PRG+HSS方法论，就得到protocol和结论：

$\mathsf{PCG.Gen}(1^\lambda)$:

- **Generate the encryption keys.** Generate keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$. Choose $\mathsf{sk}_0 \xleftarrow{\$} \mathcal{R}_q^\kappa$ and set $\mathsf{sk}_1 := \mathsf{sk} - \mathsf{sk}_0$.
- **Choose and encrypt a PRG-seed.** Choose $r \leftarrow \mathcal{D}^\ell(\mathcal{R}_p)$. Compute

$$\boldsymbol{c}^r = \mathsf{PKE.Enc}(\mathsf{pk}, r) \in (\mathcal{R}_q^\kappa)^\ell.$$

- Output $\mathsf{k}_0 := (\mathsf{sk}_0, \boldsymbol{c}^r)$, $\mathsf{k}_1 := (\mathsf{sk}_1, \boldsymbol{c}^r)$.

$\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma, f)$:

- Parse $\mathsf{k}_\sigma =: (\mathsf{sk}_\sigma, \boldsymbol{c}^r)$.
- **Evaluate $f \circ \mathsf{PRG}$ on the encrypted seed.** Compute

$$\boldsymbol{c}^Y \leftarrow \mathsf{PKE.Eval}(f \circ \mathsf{PRG}, \boldsymbol{c}^r) \in (\mathcal{R}_q^\kappa)^m.$$

- **Decrypt the result.** Decrypt and output

$$R_\sigma^Y \leftarrow \mathsf{PKE.DDec}(\mathsf{sk}_\sigma, \boldsymbol{c}^Y) \in \mathcal{R}_p^m.$$

Fig. 9. PCG for the family of degree-d functions from degree-c $\mathcal{D}^\ell$-PRG PRG and depth-$\lceil \log cd \rceil$ somewhat homomorphic encryption scheme PKE.

Theorem 55. *Let $\mathcal{R}$ be a ring, $\ell, n, p, q, m \in \mathbb{N}$, PRG be a degree-c $\mathcal{D}^\ell$-PRG $\mathsf{PRG}\colon \mathcal{R}_p^\ell \to \mathcal{R}_p^n$ and $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a depth-$\lceil \log cd \rceil$ somewhat homomorphic encryption scheme with message space $\mathcal{R}_p$ and secret key space contained in $\mathcal{R}_q^\kappa$. If PKE additionally support distributed decryption, then the PCG $\mathsf{PCG} = (\mathsf{PCG.Setup}, \mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ from Figure 9 is a PCG for the family of functions $\mathcal{F} := \{f\colon \mathcal{R}_p^n \to \mathcal{R}_p^m \mid f \text{ is of degree at most } d\}$.*

然后具体地，我们采用BGV encryption scheme [BGV12]得到：

Corollary 57. *Instantiating the PRG in the construction of Figure 9 with the degree-2 ρ-sparse PRG $\mathsf{PRG}_{\mathsf{MQ}} : \mathbb{Z}_p^\ell \to \mathbb{Z}_p^n$ from Definition 34 and the somewhat homomorphic encryption scheme with the BGV encryption scheme of Brakerski et al. [BGV12] (chosing parameters $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1), p, q$ and error distribution χ s.t. evaluation of at least degree-5 functions/ depth-3 circuits is supported), we obtain a PCG for the generation of authenticated Beaver triples, assuming ρ-sparse $\mathcal{M}(\ell^2, n, \mathcal{R}_p)$-MQ and $\mathsf{RLWE}_{N,q,\chi}$.*

效率

实际使用Beaver Triple时，我们需要一次生成一大堆Beaver triples（？）所以lattice-based很practical。

为了达到这个目的我们可以使用**ciphertext packing**技术

Remark 58 (Ciphertext packing, [SV14]). Let p be a prime and $N \in \mathbb{N}$ a power of 2, such that the polynomial $X^N + 1$ splits over $\mathbb{Z}_p$ into pairwise different degree-1 polynomials. If $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$ (similar for general cyclotomic polynomials), this implies $\mathcal{R}_p \cong (\mathbb{Z}_p)^N$ and enables "packing" N plaintexts into one ciphertext (by encrypting $\Psi(z)$ for some $z \in \mathbb{Z}_p^N$, where $\Psi\colon (\mathbb{Z}_p)^N \to \mathcal{R}_p$). In the following we will refer to $\mathcal{R}_p$ as *coefficient representation*, and to $(\mathbb{Z}_p)^N$ as *CRT representation*.

Thus, each ciphertext has room to hold N encryptions. We first consider "naive" ciphertext packing: We start with ℓ encryptions of each N seeds $r \in \mathbb{Z}_p^\ell$, perform the expansion homomorphically on the ciphertexts (which corresponds to expanding feach of the N seeds in parallel). This gives an output of nN correlated tuples in total.

最后我们分析效率（communication和computation）：

In the following we estimate efficiency of the PCG construction given in Corollary 57 with the above described ciphertext packing. We use the parameters given in [CS16] to support depth-4

homomorphic operations (as an upper bound) and plaintext space modulus $\approx 2^{128}$ listed in the following. Here, by T_M we denote the time required for multiplication over $\mathcal{R}_q$ and by T_C the time for multiplication of a $\mathbb{Z}_q$ element with an element in $\mathcal{R}_q$.

- *Dimension of $\mathcal{R}$ (over $\mathbb{Z}$):* $N \approx 13688$ (we use $N = 2^{14}$)
- *Ciphertext modulus:* $\log q \approx 750$ (we use $\log q = 744$)
- *Parameter for key switching:* $\log T \approx 140$
- *Cost of key switching:* $T_{\mathsf{KS}} \approx 2(\log q / \log T) T_C$
- *Cost of multiplication on ciphertexts:* $T_{\mathsf{Eval}} \approx 4T_M + T_{\mathsf{KS}}$
- *Cost of multiplication of constant with ciphertext:* $\approx 2T_C$
- *Cost of encryption:* $T_{\mathsf{Enc}} \approx 2(T_M + T_C)$
- *Cost of decryption:* $T_{\mathsf{Dec}} \approx 2T_M$

For MQ we use parameters $n = \ell^2/24$ and $\rho = 100$. Further, we set $\ell = c \cdot 2^9$ for $c \geq 1$. Later we will see that chosing $c = 1$ we surpass the breakeven point. In other words, $\ell = 2^9$ is the smallest choice where the total output size of the correlation generator exceeds the seed-length. Our runtime estimates are based on NFLLib [ABG+16]: A multiplication over $\mathcal{R}_q$ requires time ≈ 9.54 ms and a multiplication over $\mathcal{R}_q \times \mathbb{Z}_q$ requires time ≈ 0.55 ms. For an overview of estimated setup computation and communication complexity (i.e. time and communication required for jointly generating the seed) and estimated expansion times for the described PCG construction and variants we refer to Table 2 in the main body.

Distributed seed generation: We first describe the setup of the keys and MAC $\alpha \in \mathbb{Z}_p$, which can be reused across many instances. First, the parties jointly generate secret key shares $(\mathsf{sk}_0, \mathsf{sk}_1)$ and the corresponding public key pk, e.g. by generating secret keys according to a suitable distribution and exchanging shares as well as the corresponding public keys. Next, both parties choose a MAC share $\alpha_\sigma \xleftarrow{\$} \mathbb{Z}_p$ and define $\boldsymbol{\alpha}_\sigma \in \mathbb{Z}_p^N$ to be the vector of all α_σ entries. Next, the parties each compute and exchange $\boldsymbol{c}^{\Psi(\boldsymbol{\alpha}_\sigma)} := \mathsf{Enc}(\mathsf{pk}, \Psi(\boldsymbol{\alpha}_\sigma))$, and set $\boldsymbol{c}^{\Psi(\boldsymbol{\alpha})} := \boldsymbol{c}^{\Psi(\boldsymbol{\alpha}_0)} + \boldsymbol{c}^{\Psi(\boldsymbol{\alpha}_1)}$.

To generate encryptions of N seeds a and b in $\mathbb{Z}_p^\ell$, both parties repeat 2ℓ times: Sample an element $\mathcal{R}_p$ at random (corresponding to N random $\mathbb{Z}_p$ elements), and as for generating an encryption of the MAC key, exchange and add up the corresponding encryption.

As computation and communication is dominated by the last step, a rough estimate in the semi-honest setting are as follows: Generating 2ℓ encryptions takes about $c \cdot 20$ seconds of computation and exchanging 2ℓ ciphertexts (each of size $2N \log q$ bits) requires $c \cdot 3$ GB of communication (per party). We estimate that in the dishonest setting communication complexity would roughly double.

Expansion rate: We expand $2\ell N$ elements in $\mathbb{Z}_q$ to nN shared authenticated Beaver triples in $\mathbb{Z}_p$ (each consisting of 6 $\mathbb{Z}_p$ elements), which corresponds to expanding roughly $c \cdot 3$ GB of seed material to authenticated Beaver triples of total size $c^2 \cdot 17$ GB.

Computational efficiency of expansion: The computational costs add up as follows.

- *Expanding the seed:* The complexity to evaluate the PRG homomorphically on 2ℓ ciphertexts sums up to $2\ell^2$ ciphertext multiplications and $4n\rho$ multiplications of a constant with a ciphertext.
- *Computing the triples:* Evaluation of f_α requires $4n$ ciphertexted multiplications.
- *Obtaining the output shares:* To obtain the output we have to decrypt $n6$ ciphertexts.

Altogether, the costs sum up to

$$\approx 4n\rho T_C + 4(2\ell^2 + 7n)T_M + 2(\ell^2 + 2n)T_{\mathsf{KS}}.$$

This gives a total computation time of around $c^2 \cdot 8.0$ hours, which corresponds to an amortized computations time of roughly 0.16 ms per authenticated Beaver triple.

Multi-Party PCG for Bilinear Correlations

给出一类bilinear correlation的构造，可以延伸到很多种CR。

> In this section, we construct multi-party PCGs for a useful class of bilinear correlations, capturing M-party OT, M-party vector OLE, M-party Beaver triples, and more.

给出的是一种从2-party到M-party的方法，但是重要的是，要满足“programmability”性质，大意是说可以反复使用输入不减弱安全性。

> Our construction approach provides a semi-generic transformation from any PCG for the corresponding 2-party bilinear correlation that satisfies an additional “programmability” property. Roughly, this property requires a way of “reusing” inputs across instances without compromising security. The M-party construction will leverage this structure by executing $M(M-1)$ pairwise instances of the underlying 2-party PCG, for all the “cross-terms.”

特别的，我们所用的，满足这个programmability的PCG有：
– M-party VOLE: From lightweight DPF and LPN, leveraging the 2-party VOLE generator of [BCGI18].
– M-party OT / Beaver triple: From group-based or lattice-based HSS, leveraging our 2-party PCGs from the previous sections.

2-party simple bilinear correlation定义：

Definition 37 (Simple Bilinear Correlation: 2-party). *A 2-party correlation $\mathcal{C}$ is a* simple bilinear relation *if there exists Abelian groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and a bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ for which $\mathcal{C}$ is a distribution over $(\mathbb{G}_1, \mathbb{G}_T) \times (\mathbb{G}_2 \times \mathbb{G}_T)$ of the form*

$$\mathcal{C} = \{((a,c),(b,d)) \mid a \leftarrow \mathbb{G}_1, b \leftarrow \mathbb{G}_2, c \leftarrow \mathbb{G}_T, d = e(a,b) + c\}\,.$$

Note that the groups $\mathbb{G}$ and map e are implicitly parametrized by λ.

它蕴含着很重不同的CR：

This captures, for example, Vector OLE (with $\mathbb{G}_1 = \mathbb{G}_T = \mathbb{F}^n$ and $e\colon \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}^n$ by $e(\boldsymbol{u}, x) = x\boldsymbol{u}$), n-OLE (with $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}_T = \mathbb{F}^n$ and $e\colon \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}^n$ by $e(\boldsymbol{u}, \boldsymbol{y}) = \boldsymbol{u} * \boldsymbol{v}$ componentwise multiplication), and String OT and n-OT as special cases for $\mathbb{F} = \mathbb{F}_2$.

multi-party 版本定义：

Definition 38 (Simple Bilinear Correlation: M-party). *For simple bilinear 2-party correlation $\mathcal{C}_2$ specified by $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ we define the corresponding M-party correlation $\mathcal{C}_M$ by*

$$\mathcal{C}_M = \left\{ (a_i, b_i, c_i)_{i\in[M]} \;\middle|\; \begin{array}{c} a_i \xleftarrow{\$} \mathbb{G}_1, b_i \xleftarrow{\$} \mathbb{G}_2 \;\forall i \in [M],\; c_i \xleftarrow{\$} \mathbb{G}_T \;\forall i \in [M-1], \\ c_M = e\left(\sum_{i=1}^{M} a_i, \sum_{i=1}^{M} b_i\right) - \sum_{i=1}^{M-1} c_i \end{array} \right\}$$

Example 39. Useful specific examples:

- $\mathcal{C}_{M\text{-}\mathsf{VOLE}}$: Each party holds random $(\boldsymbol{u}_i, x_i, \boldsymbol{v}_i) \in \mathbb{F}^n \times \mathbb{F} \times \mathbb{F}^n$ s.t. $(\sum x_i)(\sum \boldsymbol{u}_i) = (\sum \boldsymbol{v}_i)$
- $\mathcal{C}_{M\text{-}\mathsf{OLE}}$: Each party holds random $(\boldsymbol{u}_i, \boldsymbol{v}_i, \boldsymbol{w}_i) \in \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n$ s.t. $(\sum \boldsymbol{u}_i) * (\sum \boldsymbol{v}_i) = (\sum \boldsymbol{w}_i)$ (componentwise)

Programmability性质的定义：

Definition 40 (Programmability). *We will say that a PCG* $\mathsf{PCG} = (\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *for simple bilinear 2-party correlation $\mathcal{C}_2$ (specified by $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$) support* reusable inputs *if* $\mathsf{PCG.Gen}(1^\lambda)$ *takes additional random inputs $a', b' \in \{0,1\}^\star$ such that:*

- ***Programmability.*** *There exist public efficiently computable functions f_a, f_b for which*

$$\Pr\left[\begin{array}{l} a', b' \leftarrow \$, (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda, a', b') \\ (a, c) \leftarrow \mathsf{PCG.Expand}(0, \mathsf{k}_0), \\ (b, d) \leftarrow \mathsf{PCG.Expand}(1, \mathsf{k}_1) \end{array} : \begin{array}{l} a = f_a(a') \\ b = f_b(b') \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

- ***Security.*** *The distributions*

$$\left\{ (\mathsf{k}_1, b', f_a(a')) \;\middle|\; \begin{array}{l} (a', b') \leftarrow \$ \\ (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda, a', b') \end{array} \right\} \quad \textit{and}$$

$$\left\{ (\mathsf{k}_1, b', f_a(\tilde{a})) \;\middle|\; \begin{array}{l} (a', b') \leftarrow \$, \tilde{a} \leftarrow \$ \\ (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda, a', b') \end{array} \right\}$$

are computationally close. A symmetric requirement holds for $b', \tilde{b}$.

最终结论：

Theorem 41 (Multi-party Simple Bilinear PCG). *Let* $\mathsf{PCG}_2 = (\mathsf{PCG}_2.\mathsf{Gen}, \mathsf{PCG}_2.\mathsf{Expand})$ *be a programmable PCG for simple bilinear 2-party correlation* $\mathcal{C}_2$ *(specified by* $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$*) with key sizes* $s_0(\lambda), s_1(\lambda)$*. Then there exists a PCG* $\mathsf{PCG}_M = (\mathsf{PCG}_M.\mathsf{Gen}, \mathsf{PCG}_M.\mathsf{Expand})$ *for the corresponding* M*-party correlation* $\mathcal{C}_M$ *with the following properties.*

- $\mathsf{PCG}_M.\mathsf{Gen}(1^\lambda)$ *runs* $M(M-1)$ *executions of* $\mathsf{PCG}_2.\mathsf{Gen}$*; each output key* k_i*,* $i \in [M]$*, has size* $(M-1)(s_0(\lambda) + s_1(\lambda) + \lambda)$ *bits.*
- $\mathsf{PCG}_M.\mathsf{Expand}(i, \mathsf{k}_i)$ *runs* $2(M-1)$ *executions of* $\mathsf{PCG}_2.\mathsf{Expand}$ *and makes* $(M-1)$ *evaluations of a pseudorandom generator.*

该定理的证明：

Proof. We analyze the following M-party PCG construction:

- $\mathsf{PCG}_M.\mathsf{Gen}(1^\lambda)$:
  1. Sample random $a'_1, \ldots, a'_M \xleftarrow{\$} \{0,1\}^\lambda$, $b'_1, \ldots, b'_M \xleftarrow{\$} \{0,1\}^\lambda$ as specified by programmability property.
  2. For every $i \neq j \in [M]$: Run $\mathsf{k}_0^{ij}, \mathsf{k}_1^{ij} \leftarrow \mathsf{PCG}_2.\mathsf{Gen}(1^\lambda, a'_i, b'_j)$ and sample PRG seed $s^{ij} \xleftarrow{\$} \{0,1\}^\lambda$
  3. For each $i \in [M]$, output $\mathsf{k}_i = \left(\{s^{ij}\}_{j\neq i}, \{\mathsf{k}_0^{ij}\}_{j\neq i}, \{\mathsf{k}_1^{ji}\}_{j\neq i}\right)$
- $\mathsf{PCG}_M.\mathsf{Expand}(i, \mathsf{k}_i)$:
  1. For every $j \neq i$, compute

$$r_{ij} \leftarrow \mathsf{PRG}(s^{ij}),$$
$$(a_{ij}, c_{ij}) \leftarrow \mathsf{Expand}_2(0, \mathsf{k}_0^{ij}), (b_{ji}, d_{ji}) \leftarrow \mathsf{Expand}_2(1, \mathsf{k}_1^{ji})$$

  2. Output $A_i = a_{ij}$, $B_i = b_{ji}$ (same for all j) and
     $C_i = -\sum_{j\neq i} c_{ij} + \sum_{j\neq i} d_{ji} + e(A_i, B_i) + (-1)^{[i<j]} r_{ij}$,
     where $[i < j] = 1$ if $i < j$ and 0 if $j < i$.

Correctness: By assumption, each expanded output from $(\mathsf{PCG}_2.\mathsf{Gen}, \mathsf{PCG}_2.\mathsf{Expand})$ is computationally indistinguishable from $\mathcal{C}_2$. In particular, it must hold whp for all $i \neq j$: $e(A_i, B_j) = d_{ij} - c_{ij}$, where $(\mathsf{k}_0^{ij}, \mathsf{k}_1^{ij}) \leftarrow \mathsf{PCG}_2.\mathsf{Gen}(1^\lambda, a'_i, b'_j), (a_{ij}, c_{ij}) \leftarrow \mathsf{PCG}_2.\mathsf{Expand}(0, \mathsf{k}_0^{ij}), (b_{ij}, d_{ij} \leftarrow$

$\mathsf{PCG}_2.\mathsf{Expand}(1, \mathsf{k}_1^{ij})$. This means with overwhelming probability,

$$\begin{aligned} e\left(\sum_{i=1}^{M} A_i, \sum_{j=1}^{M} B_j\right) &= \sum_{i=1}^{M}\sum_{j=1}^{M} e\left(A_i, B_j\right) \\ &= \sum_{i=1}^{M} e\left(A_i, B_i\right) + \sum_{i=1}^{M}\sum_{j\neq i} e\left(A_i, B_j\right) \\ &= \sum_{i=1}^{M} e\left(A_i, B_i\right) + \sum_{i=1}^{M}\sum_{j\neq i} (d_{ij} - c_{ij}) = \sum_{i=1}^{k} C_i. \end{aligned}$$

(Note that for all $i \neq j$, r_{ij} is added and subtracted exactly once in the sum.) Further, by indistinguishability of the expanded outputs from $\mathsf{PCG}_2.\mathsf{Expand}$ to the target correlation $\mathcal{C}_2$, it holds that each (a_i, b_j) pair is pseudorandom. Thus, for M independent samples, $\sum a_i$ and $\sum b_i$ are jointly pseudorandom. Finally, from the pairwise pseudorandom offsets r_{ij} (independent of the a_i and b_i), it holds that the C_i are pseudorandom, up to the required constraint. Correctness follows.

Security. We now proceed to prove security of PCG_M. Let $T \subset [M]$ corrupted. We wish to show that given $\{\mathsf{k}_i\}_{i\in T}$, the expanded outputs of honest parties $(A_i, B_i, C_i)_{i\notin T}$ cannot be distinguished from an independent resampling, conditioned on the *expanded* values $(A_i, B_i, C_i)_{i\in T}$ of the corrupt seeds.

We first observe that due to the pariwise secret pseudorandom offsets $r^{ij} = \mathsf{PRG}(s^{ij})$, that even given $\{\mathsf{k}_i\}_{i\in T}$ and $(A_i, B_i)_{i\in T}$ the joint distribution of $(C_i)_{i\notin T}$ is indistinguishable from random, up to the preserved sum $\sum_{i\notin T} C_i$ as required.

It thus remains to show that given $\{\mathsf{k}_i\}_{i\in T}$, the expanded honest values $(A_i, B_i)_{i\notin T}$ are pseudorandom. By a hybrid argument, we may replace the values of honest A_i and B_i one at a time. It then suffices to adress an extreme case of this step, where all but one party $i \in [M]$ is corrupted.

We first treat A_i; the argument for B_i is symmetric. For any $i \in [M]$,

$$\left\{ (\{\mathsf{k}_j\}_{j\neq i}, (A_i, B_i)) \;\middle|\; \begin{array}{l} (\mathsf{k}_1, \ldots, \mathsf{k}_M) \xleftarrow{\$} \mathsf{PCG}_M.\mathsf{Gen}(1^\lambda) \\ (A_i, B_i, C_i) \xleftarrow{\$} \mathsf{PCG}_M.\mathsf{Expand}(i, \mathsf{k}_i) \end{array} \right\}$$

$$\equiv \left\{ \left(\{\mathsf{k}_1^{ij}\}_{j\neq i}, f_a(a_i'), X\right) \;\middle|\; \begin{array}{l} a_i' \leftarrow \$, b_j' \leftarrow \$\ \forall j \neq i \\ (\mathsf{k}_0^{ij}, \mathsf{k}_1^{ij}) \leftarrow \mathsf{PKE}_2.\mathsf{Gen}(1^\lambda, a_i', b_j') \\ X \leftarrow \mathsf{RestOfSeeds}(\{b_j'\}_{j\neq i}) \end{array} \right\},$$

where $\mathsf{RestOfSeeds}$ is an efficiently sampleable distribution that samples $b_i' \leftarrow \$, a_j' \leftarrow \$\ \forall j \neq i$, executes the remaining $(2M-1)(M-1)$ instances of $\mathsf{PCG}_2.\mathsf{Gen}(1^\lambda, a_\ell', b_j')$ and outputs

$$\left(\{\mathsf{k}_0^{j\ell}\}_{j\neq i, l\in[M]}, \{\mathsf{k}_1^{\ell j}\}_{j\neq i, l\neq i}, B_i = f_b(b_i')\right).$$

By a direct sequence of $(M-1)$ hybrids over $j \neq i \in [M]$ we may appeal to the security of $(\mathsf{PCG}_2.\mathsf{Gen}, \mathsf{PCG}_2.\mathsf{Expand})$ to iteratively replace the jth key k_1^{ij} generated by $(\mathsf{k}_0^{ij}, \mathsf{k}_1^{ij}) \leftarrow \mathsf{PCG}_2.\mathsf{Gen}(1^\lambda, a_i', b_j')$ with $\tilde{\mathsf{k}}_1^{ij}$ generated as $(\tilde{\mathsf{k}}_0^{ij}, \tilde{\mathsf{k}}_1^{ij}) \leftarrow \mathsf{PCG}_2.\mathsf{Gen}(1^\lambda, \tilde{a}_i, b_j')$, for independent $\tilde{a}_i \leftarrow \$$.

We thus obtain the above distribution is indistinguishable from

$$\approx \left\{ \left(\{\tilde{\mathsf{k}}_1^{ij}\}_{j\neq i}, f_a(a'_i), X \right) \middle| \begin{array}{l} a'_i \leftarrow \$, \tilde{a}_i \leftarrow \$, b'_j \leftarrow \$ \; \forall j \neq i \\ (\mathsf{k}_0^{ij}, \mathsf{k}_1^{ij}) \leftarrow \mathsf{PKE}_2.\mathsf{Gen}(1^\lambda, a'_i, b'_j) \\ X \leftarrow \mathsf{RestOfSeeds}(\{b'_j\}_{j\neq i}) \end{array} \right\}.$$

However, in this case $A_i = f_a(a'_i)$ for $a'_i \leftarrow \$$ is completely independent of $\left(\{\tilde{\mathsf{k}}_1^{ij}\}_{j\neq i}, X\right)$, generated now as a function in only $\tilde{a}_i$ (not a'_i). The claim, and thus security of the construction, follows.

有什么2-party PCG满足programmability? ：

Proposition 59 (Programmability of 2-party PCGs). *The following 2-party PCGs are* programmable*, as per Definition 40.*

- *The 2-party VOLE generator of [BCGI18], based on DPF and LPN.*
- *The PCGs for arbitrary simple 2-party bilinear correlations as constructed in this work from somewhat-homomorphic encryption or BGN.*

Proof. M-party VOLE. Recall the 2-party VOLE generator of [BCGI18] takes the following form (we describe their "dual" construction). The sender party receives a (short representation of) a sparse random vector $\boldsymbol{y}$ over the field $\mathbb{F}$, the receiver receives a field element $x \in \mathbb{F}$, and each receives an FSS share of a multi-point function corresponding to the product $x\boldsymbol{y}$. The scheme is parameterized by a public matrix H for which the dual-LPN problem is hard (with respect to sparse noise). The sender expands to output $(\boldsymbol{u}, \boldsymbol{v})$, and the receiver to $(x, \boldsymbol{w})$.

It was already observed in [BCGI18] that the construction was programmable with respect to the receiver's output x. We observe that a similar programmability holds also for the output $\boldsymbol{u}$ of the sender, where in particular $\boldsymbol{u}$ is the output of compressing the value $\boldsymbol{y}$ via the public matrix H. In both cases, the "programming information" ($a' = \boldsymbol{u}$ and $b' = x$) is anyway given to the respective party, so the required security notion is directly implied by standard PCG security.

General Simple Bilinear via HSS. One can support 2-party PCG of any simple bilinear correlation via our PCGs for degree-2 correlations (e.g., obtained from lattices and BGN) by giving each party a short seed a', b', respectively, as well as HSS shares of a' and b' that support homomorphic evaluation of individual PRG expansion $a = \mathsf{PRG}(a'), b = \mathsf{PRG}(b')$ and then the multiplication ab. This construction inherently supports programmability, by the initial values a', b'.