Implementing Candidate Graded Encoding Schemes from Ideal Lattices

Martin R. Albrecht [1], Catalin Cocis [2], Fabien Laguillaumie [3] and **Adeline Langlois** [4]

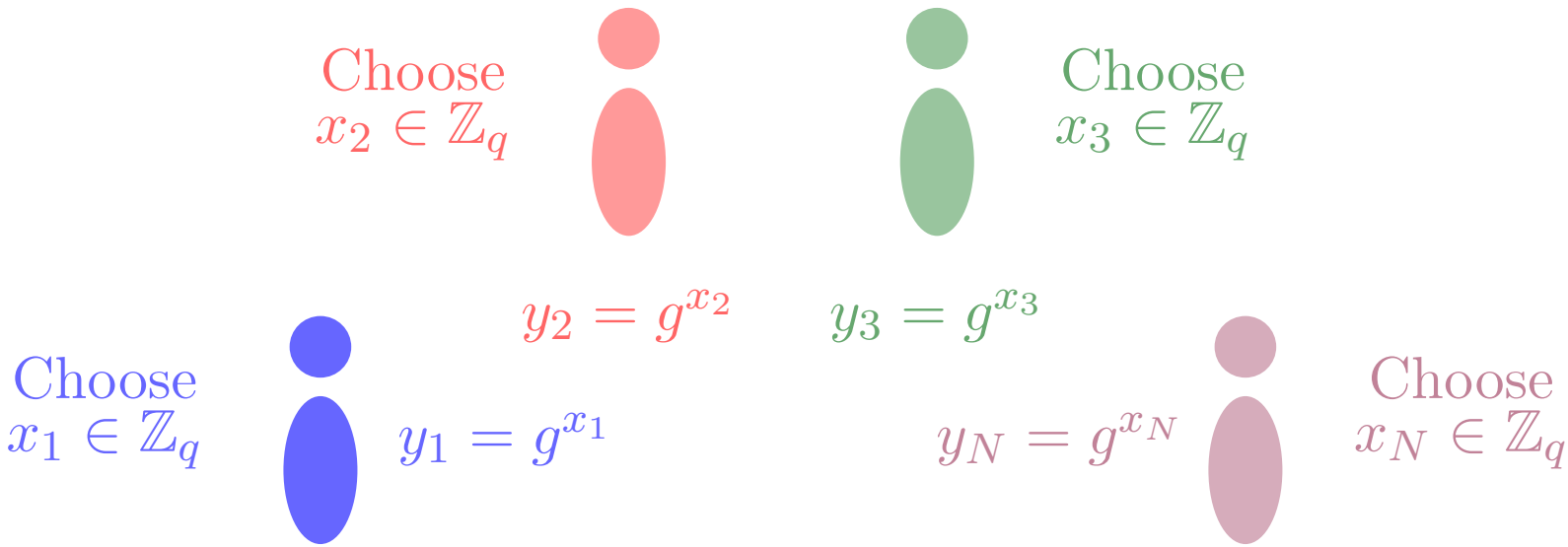
1. Information Security Group, Royal Holloway, University of London
2. Technical University of Cluj-Napoca
3. UCBL Lyon 1 (U. Lyon, CNRS, ENS Lyon, INRIA, UCBL)
4. EPFL, Lausanne, Switzerland and CNRS/IRISA, Rennes, France

December 3, 2015

Cryptographic Multilinear Maps

Group of $N > 2$ parties want to communicate privately via cloud.

$\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ with q prime, g public generator of $\mathbb{Z}_q^\times$



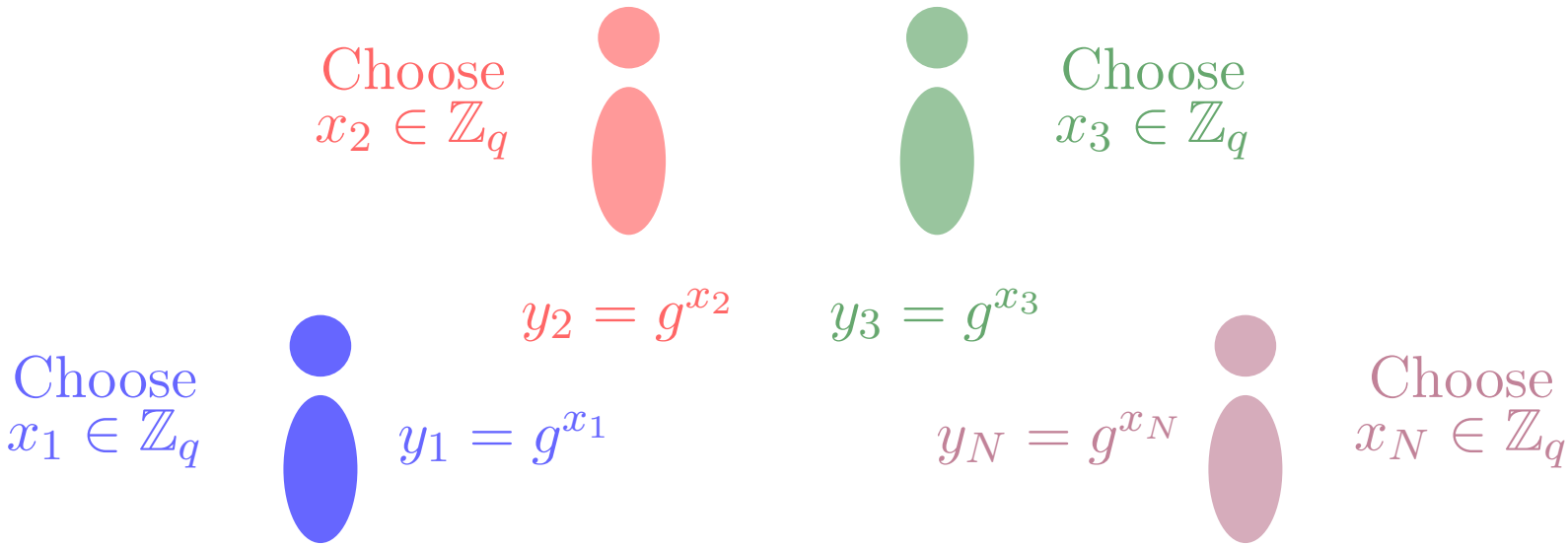
Secret key (using e: "cryptographic multilinear map"):

$$K = e(g, \ldots, g)^{x_1 \cdots x_N} = e(y_2, y_3, \ldots, y_N)^{x_1} = e(y_1, y_3, \ldots, y_N)^{x_2}$$

Cryptographic Multilinear Maps

Group of $N > 2$ parties want to communicate privately via cloud.

$\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ with q prime, g public generator of $\mathbb{Z}_q^\times$



Secret key (using e: "cryptographic multilinear map"):

$$K = e(g, \ldots, g)^{x_1 \cdots x_N} \quad = e(y_2, y_3, \ldots, y_N)^{x_1}$$

- Security: Hardness of **Multilinear Decisional DH** problem,

  MDDH: For $x_1, \ldots, x_N, x' \leftarrow U(\mathbb{Z}_q)$, distinguish between $(g^{x_1}, \ldots, g^{x_N}, e(g, \ldots, g)^{x_1 \cdots x_N})$ and $(g^{x_1}, \ldots, g^{x_N}, e(g, \ldots, g)^{x'})$.

Construction?

For $N = 3$ use bilinear maps

$e : G_1 \times G_2 \to G_T$ and $g_1 \in G_1$, $g_2 \in G_2$, $g_T \in G_T$ generators.

- $e(\cdot, \cdot)$ is bilinear: $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$,
- $e(\cdot, \cdot)$ is non-degenerate: $e(g_1, g_2)$ generates G_T,
- $e(\cdot, \cdot)$ efficiently computable and DLOG hard in all groups.

Construction?

For $N = 3$ use bilinear maps

$e : G_1 \times G_2 \to G_T$ and $g_1 \in G_1$, $g_2 \in G_2$, $g_T \in G_T$ generators.

- $e(\cdot, \cdot)$ is bilinear: $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$,
- $e(\cdot, \cdot)$ is non-degenerate: $e(g_1, g_2)$ generates G_T,
- $e(\cdot, \cdot)$ efficiently computable and DLOG hard in all groups.

Ideal construction of cryptographic multilinear map (extend this to κ elements) does not exist.

Construction?

Ideal construction of cryptographic multilinear map (extend this to κ elements) does not exist.

Approximation: Graded Encoding Scheme

Think of

x	as a "level-0" encoding of x,
g^x	as a "level-1" encoding of y,
$e(g,g)^{xy}$	as a "level-2" encoding of xy,
$e(\cdot,\ldots,\cdot)$	as "multiplying" two elements at level i and j to produce an element at level $i+j$,
$g^x \cdot g^y$	as "adding" two elements at the same level.

Cryptographic Multilinear Maps – History

- 2000: 3-parties key agreement using pairings [Joux00]
- 2003: $\kappa + 1$-parties using κ-linear maps [BonehSilverberg 2003]

What happenned in the last three years?

- 2012: First plausible realization [GargGentryHalevi 2013]
  - New applications: indistinguishablily obfuscation (iO)
- 2013: Variant over the integers [CoronLepointTibouchi 2013]
- 2014: Graph-induced Mmaps [GentryGorbunovHalevi 2015]

Cryptographic Multilinear Maps – History

- 2000: 3-parties key agreement using pairings [Joux00]
- 2003: $\kappa + 1$-parties using κ-linear maps [BonehSilverberg 2003]

What happenned in the last three years?

- 2012: First plausible realization [GargGentryHalevi 2013]
  - New applications: indistinguishablily obfuscation (iO)
  - **Attacked** by [HuJia 2015]
- 2013: Variant over the integers [CoronLepointTibouchi 2013]
  - **Attacked** by [CheonHanLeeRyuStehlé 2014]
  - Fixed in [CoronLepointTibouchi 2015]
  - Fix **fully broken** [CheonLeeRyu 2015] [MinaudFouque 2015]
- 2014: Graph-induced Mmaps [GentryGorbunovHalevi 2015]
  - Recently **attacked** by [Coron 2015]

GGH13 graded encoding scheme

- In bilinear map (g and e public):
  anyone can "encode": given a secret x, compute g^x,
  given g^{x_1}, g^{x_2} and secret x_3, compute $e(g^{x_1}, g^{x_2})^{x_3}$.

- In graded encoding schemes, two possible versions:

  - A "secret key" version:
    Only the person who have the secret can encode,
    Application: indistinguishability obfuscation (iO).

  - A "public key" version:
    Publish some public elements then anyone can encode,
    Possible application: multi-parties key exchange.

GGH: two versions - "secret key version"

$\mathcal{I} = (g)$ prime ideal over $R(= \mathbb{Z}[x]/(x^n + 1))$ with small g (secret), $R_{\text{Enc}} = R_q$ and $R_{\text{Plain}} = R/(g)$, κ is the degree of multilinearity

- **Plaintext:** e element of $R/(g)$,
- **Level-1 encoding:** $[c/z]_q$ for $z \leftarrow U(R_q)$ (secret).
  - where c is a small coset representative of $e + (g)$,
- **Level-k encoding:** $[c/z^k]_q$

GGH: two versions - "secret key version"

$\mathcal{I} = (g)$ prime ideal over $R(= \mathbb{Z}[x]/(x^n + 1))$ with small g (secret), $R_{\text{Enc}} = R_q$ and $R_{\text{Plain}} = R/(g)$, κ is the degree of multilinearity

- **Plaintext:** e element of $R/(g)$,
- **Level-1 encoding:** $[c/z]_q$ for $z \leftarrow U(R_q)$ (secret).
  - where c is a small coset representative of $e + (g)$,
- **Level-k encoding:** $[c/z^k]_q$
- **Adding encodings add:** Given $u_1 = [c_1/z^k]_q$ and $u_2 = [c_2/z^k]_q$:
  - $u = [u_1 + u_2]_q = [(c_1 + c_2)/z^k]_q$ is a level-k encoding of $[c_1 + c_2]_g$.
- **Multiplying enc mult:** Given $u_1 = [c_1/z^{k_1}]_q$, $u_2 = [c_2/z^{k_2}]_q$:
  - $u = [u_1 \cdot u_2]_q = [(c_1 \cdot c_2)/z^{k_1+k_2}]_q$: level-$(k_1 + k_2)$ enc of $[c_1 \cdot c_2]_g$.

GGH: two versions - "secret key version"

$\mathcal{I} = (g)$ prime ideal over $R(= \mathbb{Z}[x]/(x^n + 1))$ with small g (secret), $R_{\text{Enc}} = R_q$ and $R_{\text{Plain}} = R/(g)$, κ is the degree of multilinearity

- **Plaintext:** e element of $R/(g)$,
- **Level-1 encoding:** $[c/z]_q$ for $z \leftarrow U(R_q)$ (secret).
  - where c is a small coset representative of $e + (g)$,
- **Level-k encoding:** $[c/z^k]_q$
- **Adding encodings** add: Given $u_1 = [c_1/z^k]_q$ and $u_2 = [c_2/z^k]_q$:
  - $u = [u_1 + u_2]_q = [(c_1 + c_2)/z^k]_q$ is a level-k encoding of $[c_1 + c_2]_g$.
- **Multiplying enc** mult: Given $u_1 = [c_1/z^{k_1}]_q$, $u_2 = [c_2/z^{k_2}]_q$:
  - $u = [u_1 \cdot u_2]_q = [(c_1 \cdot c_2)/z^{k_1+k_2}]_q$: level-$(k_1 + k_2)$ enc of $[c_1 \cdot c_2]_g$.
- **Zero-testing** isZero: public parameter: $p_{zt} = [\frac{h}{g} z^\kappa]_q$ with "small" h, Given $u = [c/z^\kappa]_q$, return 1 if $\|[p_{zt} \cdot u]_q\|_\infty \leq q^{3/4}$.
  - $[p_{zt} \cdot u]_q = [\frac{h}{g} z^\kappa \cdot c/z^\kappa]_q = [\frac{h \cdot c}{g}]_q$, small only if $c \in (g)$.

GGH: two versions - "public key version"

$\mathcal{I} = (g)$ prime ideal over $R(= \mathbb{Z}[x]/(x^n + 1))$ with small g (secret),
$R_{\text{Enc}} = R_q$ and $R_{\text{Plain}} = R/(g)$, κ is the degree of multilinearity

- **Plaintext:** e element of $R/(g)$,
- **Level-1 encoding:** $[c/z]_q$ for $z \leftarrow U(R_q)$ (secret).
  - where c is a small coset representative of $e + (g)$,
- **Level-k encoding:** $[c/z^k]_q$

GGH: two versions - "public key version"

$\mathcal{I} = (g)$ prime ideal over $R(= \mathbb{Z}[x]/(x^n + 1))$ with small g (secret), $R_{\text{Enc}} = R_q$ and $R_{\text{Plain}} = R/(g)$, κ is the degree of multilinearity

- **Public parameter:** y level-1 encoding of 1,
- **Plaintext:** e element of $R/(g)$,
- **Level-1 encoding:** $[c/z]_q = [e \cdot y]_q$ for $z \leftarrow U(R_q)$ (secret).
  - where c is a small coset representative of $e + (g)$,
- **Level-k encoding:** $[c/z^k]_q = [e \cdot y^k]_q$

GGH: two versions - "public key version"

$\mathcal{I} = (g)$ prime ideal over $R(= \mathbb{Z}[x]/(x^n + 1))$ with small g (secret), $R_{\text{Enc}} = R_q$ and $R_{\text{Plain}} = R/(g)$, κ is the degree of multilinearity

- **Public parameter:** y level-1 encoding of 1,
- **Plaintext:** e element of $R/(g)$,
- **Level-1 encoding:** $[c/z]_q = [e \cdot y]_q$ for $z \leftarrow U(R_q)$ (secret).
  - where c is a small coset representative of $e + (g)$,
- **Level-k encoding:** $[c/z^k]_q = [e \cdot y^k]_q$

To ensure security $\Rightarrow$ need randomization of the encodings

- Public parameters $\{x_j\}_{j \in [m_r]}$ level-1 encodings of zero.
- **Level-1 encoding:** $[u' + \sum_j \rho_j x_j]_q$,
  - where ρ_j is sampled from a discrete Gaussian over $\mathbb{Z}$,
  - $\sum_j \rho_j x_j$ is a discrete Gaussian and an encoding of zero.

GGH: two versions

Secret key version

- z secret used to encode
- no need of re-randomizers
- zero-testing parameter public
- Main application: indistinguishable Obfuscation

What we implement

Public key version

- y public used to encode
  $\Rightarrow$ anyone can encode
- need of "re-randomizers": level-i encodings of zero
- zero-testing parameter public
- Used for N-party key exchange

GGH: two versions

Secret key version

- z secret used to encode
- no need of re-randomizers
- zero-testing parameter public
- Main application: indistinguishable Obfuscation

What we implement

Public key version

- y public used to encode
  $\Rightarrow$ anyone can encode
- need of "re-randomizers": level-i encodings of zero
- zero-testing parameter public
- Used for N-party key exchange

using

All existing constructions are broken

Could this be implemented?

- Original GGH construction:
  parameters too big: **nothing can run in practice**.

- GGHLite has nicer parameters but still some issues:
  [LangloisStehléSteinfeld 2014]
  - (g) needs to be a prime ideal,
  - Very large parameters n and q,
  - No discrete gaussian sampling over arbitrary ideals publicly available.

Our work

First and efficient implementation of improved GGH scheme ("secret key version") publicly available

- We show that (g) does not need to be a prime ideal,
- We provide a better analysis of the scheme:
  - reduce bitsize of q by factor 4 (and then size of n),
- We give a strategy to choose efficient parameters,
  - based on lattice attacks.

Our work

First and efficient implementation of improved GGH scheme ("secret key version") publicly available

In the scheme, all operations are in $R = \mathbb{Z}[x]/(x^n + 1)$ or R_q

- Implementation in C relies on FLINT,
  with all steps in quasi-linear time,
  - Re-implement most of the non-trivial operations
    - Polynomial multiplication in R_q using NTT,
    - Computing norms in R,
  - Implement operations not available in FLINT
    - Approximate inverse in $K = \mathbb{Q}[x]/(x^n + 1)$,
    - Approximate square root in K,
    - Sampling from Discrete Gaussians on arbitrary ideals (using [GPV08,DDLL13]).
- Implementation ready to be used for implementing iO.

Some concrete results

λ	κ	λ'	n	$\log q$	Setup	Encode	Mult	$\|\text{enc}\|$
52	6	64.4	2^{15}	2117	114s	26s	0.05s	8.3MB
52	52	62.7	2^{18}	19898	26695s	1016s	84.1s	621.8MB
80	6	155.2	2^{16}	2289	415s	74s	0.13s	17.9MB
80	19	80.4	2^{17}	7089	1821s	268s	3.07s	110.8MB
80	38	80.3	2^{18}	14649	20381s	947s	16.21s	457.8MB

- κ is the multilinearity level,
- λ' expected security level based on best known attacks,
- Setup: time for generating GGH instance,
- Encode: time to reduce an element $\in \mathbb{Z}_p$ with $p = \mathcal{N}(\mathcal{I})$ to a small element in $\mathbb{Z}[X]/(x^n + 1)$ modulo (g),
- Mult lists the time to multiply κ elements.

Conclusion

Implementing lattice-based schemes (in $R = \mathbb{Z}[x]/(x^n + 1)$)

Part of this implementation may be useful and will be soon be available independently.

Open problems

Security of graded encoding schemes:

- Attacking the "secret key" variant of GGH or CLT,
- Constructing a secure variant.

Conclusion

Implementing lattice-based schemes (in $R = \mathbb{Z}[x]/(x^n + 1)$)

Part of this implementation may be useful and will be soon be available independently.

Open problems

Security of graded encoding schemes:

- Attacking the "secret key" variant of GGH or CLT,
- Constructing a secure variant.

`https://bitbucket.org/malb/gghlite-flint`

Thank You