DAGS - KEY ENCAPSULATION FROM DYADIC GS CODES

Gustavo Banegas[1], Paulo S. L. M. Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiécoumba Gueye, Richard Haeussler, Jean Belo Klamti, Ousmane N'diaye, Duc Tri Nguyen, Edoardo Persichetti and Jefferson E. Ricardini

https://www.dags-project.org

Technische Universiteit **Eindhoven** University of Technology

March 20, 2018

[1]Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
`gustavo@cryptme.in`

Structured Codes

DAGS - KEM

Code-based cryptography

- McEliece: first cryptosystem using error correcting codes (1978); Based on the hardness of decoding random linear codes.

Computational Syndrome Decoding

Given: $H \in \mathbb{F}_q^{(n-k)\times n}$, $y \in \mathbb{F}_q^{(n-k)}$ and $\omega \in \mathbb{N}$.
Goal: find a word $e \in \mathbb{F}_q^n$ with $wt(e) \leq \omega$ such that $He^T = y$.

Code-based cryptography

- McEliece: first cryptosystem using error correcting codes (1978); Based on the hardness of decoding random linear codes.

Computational Syndrome Decoding

Given: $H \in \mathbb{F}_q^{(n-k)\times n}$, $y \in \mathbb{F}_q^{(n-k)}$ and $\omega \in \mathbb{N}$.
Goal: find a word $e \in \mathbb{F}_q^n$ with $wt(e) \leq \omega$ such that $He^T = y$.

- Unique solution and hardness only if ω is below a certain threshold (GV bound).

"fast" Code-based cryptography 101

Key Generation:

- Choose ω-error correcting code $\mathcal{C}$;
- SK: code description Δ for $\mathcal{C}$;
- PK: generator matrix G in systematic form for $\mathcal{C}$.

"fast" Code-based cryptography 101

Key Generation:

- Choose ω-error correcting code $\mathcal{C}$;
- SK: code description Δ for $\mathcal{C}$;
- PK: generator matrix G in systematic form for $\mathcal{C}$.

Encryption:

- Message is a word $m \in \mathbb{F}_{q^m}^k$;
- Select random error vector $e \in \mathbb{F}_{q^m}^n$ of weight ω;
- $c = mG + e$.

“fast” Code-based cryptography 101

Key Generation:

- Choose ω-error correcting code $\mathcal{C}$;
- SK: code description Δ for $\mathcal{C}$;
- PK: generator matrix G in systematic form for $\mathcal{C}$.

Encryption:

- Message is a word $m \in \mathbb{F}_{q^m}^k$;
- Select random error vector $e \in \mathbb{F}_{q^m}^n$ of weight ω;
- $c = mG + e$.

Decryption:

- Set $m =$Decode(c) and return m;
- Return “fail” if decoding fails.

Structured Codes

Structured Codes

- **Generalized Srivastava**;
- Quasi-cyclic codes (QC);
- Quasi-dyadic codes (QD);
- Quasi-Dyadic + Goppa;
- Goppa codes;
- Others. . .

Structured Codes

Structured Codes

Quasi-Dyadic Codes (Misoczki, Barreto '09).

Structured Codes

Structured Codes

Quasi-Dyadic Codes (Misoczki, Barreto '09).

Several families have QC/QD description
GRS, Goppa, Generalized Srivastava (Persichetti'11).

Structured Codes

Structured Codes

Quasi-Dyadic Codes (Misoczki, Barreto '09).

Several families have QC/QD description
GRS, Goppa, Generalized Srivastava (Persichetti'11).

Use subfield subcode construction to encrypt in the subcode and decrypt using parent code.

Structured Codes

Structured Codes

Quasi-Dyadic Codes (Misoczki, Barreto '09).

Several families have QC/QD description
GRS, Goppa, Generalized Srivastava (Persichetti'11).

Use subfield subcode construction to encrypt in the subcode and decrypt using parent code.

Problem: extra structure = extra info for attacker.

Structured Codes

Structured Codes

Quasi-Dyadic Codes (Misoczki, Barreto '09).

Several families have QC/QD description
GRS, Goppa, Generalized Srivastava (Persichetti'11).

Use subfield subcode construction to encrypt in the subcode and decrypt using parent code.

Problem: extra structure = extra info for attacker.

Critical algebraic attack (Faugère, Otmani, Perret, Tillich '10).

Generalized Srivastava Codes

Alternant codes with non-trivial intersection with Goppa codes.
Admit parity-check which is superposition of s blocks of size $t \times n$.

Generalized Srivastava Codes

- Alternant codes with non-trivial intersection with Goppa codes.
- Admit parity-check which is superposition of s blocks of size $t \times n$.
- Each block H_ℓ has ij-th element $\frac{z_j}{(v_j - u_\ell)^i}$ (nonzero field elements).
- If $t = 1$ this is a Goppa code.
- Can generate QD-GS codes using (modified) algorithm for QD Goppa.
- Solution space defined by extension degree mt.
- Similar performance, more flexibility, easier to resist FOPT ($mt > 20$).

DAGS

Select hash functions **G**, **H**,**K**;

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;
- PK: generator matrix G in systematic form for C over $\mathbb{F}_q$.

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;
- PK: generator matrix G in systematic form for C over $\mathbb{F}_q$.

Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$;

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;
- PK: generator matrix G in systematic form for C over $\mathbb{F}_q$.

Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$;
- Compute $(\rho \parallel \sigma) = \mathbf{G}(m)$, $d = \mathbf{H}(m)$ and set $\mu = (\rho \parallel m)$;

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;
- PK: generator matrix G in systematic form for C over $\mathbb{F}_q$.

Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$;
- Compute $(\rho \parallel \sigma) = \mathbf{G}(m)$, $d = \mathbf{H}(m)$ and set $\mu = (\rho \parallel m)$;
- Generate $e \in \mathbb{F}_{q^m}^n$ of weight ω from seed σ;

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;
- PK: generator matrix G in systematic form for C over $\mathbb{F}_q$.

Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$;
- Compute $(\rho \parallel \sigma) =\mathbf{G}(m)$, $d =\mathbf{H}(m)$ and set $\mu = (\rho \parallel m)$;
- Generate $e \in \mathbb{F}_{q^m}^n$ of weight ω from seed σ;
- Output (c, d) where $c = \mu G + e$ and $K =\mathbf{K}(m)$.

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;
- PK: generator matrix G in systematic form for C over $\mathbb{F}_q$.

Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$;
- Compute $(\rho \parallel \sigma) = \mathbf{G}(m)$, $d = \mathbf{H}(m)$ and set $\mu = (\rho \parallel m)$;
- Generate $e \in \mathbb{F}_{q^m}^n$ of weight ω from seed σ;
- Output (c, d) where $c = \mu G + e$ and $K = \mathbf{K}(m)$.

Decryption

- Set $(\mu', e') = Decode(c)$ and parse $\mu' = (\rho' \parallel m')$;

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;
- PK: generator matrix G in systematic form for C over $\mathbb{F}_q$.

Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$;
- Compute $(\rho \parallel \sigma) =\mathbf{G}(m)$, $d =\mathbf{H}(m)$ and set $\mu = (\rho \parallel m)$;
- Generate $e \in \mathbb{F}_{q^m}^n$ of weight ω from seed σ;
- Output (c, d) where $c = \mu G + e$ and $K =\mathbf{K}(m)$.

Decryption

- Set $(\mu', e') = \mathit{Decode}(c)$ and parse $\mu' = (\rho' \parallel m')$;
- Recompute $\mathbf{G}(m')$, $d = \mathbf{H}(m')$ and e'', then compare;

DAGS

Select hash functions **G**, **H**,**K**;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;
- PK: generator matrix G in systematic form for C over $\mathbb{F}_q$.

Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$;
- Compute $(\rho \parallel \sigma) = \mathbf{G}(m)$, $d = \mathbf{H}(m)$ and set $\mu = (\rho \parallel m)$;
- Generate $e \in \mathbb{F}_{q^m}^n$ of weight ω from seed σ;
- Output (c, d) where $c = \mu G + e$ and $K = \mathbf{K}(m)$.

Decryption

- Set $(\mu', e') = \mathit{Decode}(c)$ and parse $\mu' = (\rho' \parallel m')$;
- Recompute $\mathbf{G}(m')$, $d = \mathbf{H}(m')$ and e'', then compare;
- Return $\perp$ if decoding fails or any check fails;

DAGS

Select hash functions $\mathbf{G}$, $\mathbf{H}$,$\mathbf{K}$;

Key Generation

- Generate a QD-GS code C;
- SK: description for C (in alternant form) over $\mathbb{F}_{q^m}$;
- PK: generator matrix G in systematic form for C over $\mathbb{F}_q$.

Encapsulation

- Choose random word $m \in \mathbb{F}_q^k$;
- Compute $(\rho \parallel \sigma) = \mathbf{G}(m)$, $d = \mathbf{H}(m)$ and set $\mu = (\rho \parallel m)$;
- Generate $e \in \mathbb{F}_{q^m}^n$ of weight ω from seed σ;
- Output (c, d) where $c = \mu G + e$ and $K = \mathbf{K}(m)$.

Decryption

- Set $(\mu', e') = \mathit{Decode}(c)$ and parse $\mu' = (\rho' \parallel m')$;
- Recompute $\mathbf{G}(m')$, $d = \mathbf{H}(m')$ and e'', then compare;
- Return $\perp$ if decoding fails or any check fails;
- Else return $K = \mathbf{K}(m')$.

About DAGS

Uses McEliece and generic KEM paradigm (Hofheinz, Hövelmanns, Kiltz '17).

About DAGS

Uses McEliece and generic KEM paradigm (Hofheinz, Hövelmanns, Kiltz '17).

Leverage "randomized" IND-CPA McEliece variant for tighter security proof.

About DAGS

Uses McEliece and generic KEM paradigm (Hofheinz, Hövelmanns, Kiltz '17).
Leverage "randomized" IND-CPA McEliece variant for tighter security proof.
Efficient "Key Confirmation + Re-encryption" step.

Typical parameters:

q	m	n	k	s	t	Errors	PK Size (bytes)	SK Size (bytes)	Cipher text (bytes)
2^6	2	2112	704	2^6	11	352	11,616	6,336	1,616

Advantages: small keys and ciphertext. Disadvantages: Conservative parameters that makes DAGS slow.
Code at: https://git.dags-project.org/dags/dags

About DAGS implementation

Key generation

- Operations in $\mathbb{F}_{2^{12}}$ and in $\mathbb{F}_{2^6}$;

About DAGS implementation

Key generation

- Operations in $\mathbb{F}_{2^{12}}$ and in $\mathbb{F}_{2^6}$;
  - Additions are "cheap";
  - Multiplications and inversions are costly; **Originally with** log **and i**-log **tables**
- Random generation of a polynomial in $\mathbb{F}_{2^{12}}$.

About DAGS implementation

Key generation

- Operations in $\mathbb{F}_{2^{12}}$ and in $\mathbb{F}_{2^6}$;
  - Additions are "cheap";
  - Multiplications and inversions are costly; **Originally with** log **and i**-log **tables**
- Random generation of a polynomial in $\mathbb{F}_{2^{12}}$.

Encapsulation

- Operations in $\mathbb{F}_{2^{12}}$ and in $\mathbb{F}_{2^6}$;
- Random generation of a polynomial in $\mathbb{F}_{2^{12}}$;
- Hash function "call".

About DAGS implementation

Key generation

- Operations in $\mathbb{F}_{2^{12}}$ and in $\mathbb{F}_{2^6}$;
  - Additions are "cheap";
  - Multiplications and inversions are costly; **Originally with** log **and i**-log **tables**
- Random generation of a polynomial in $\mathbb{F}_{2^{12}}$.

Encapsulation

- Operations in $\mathbb{F}_{2^{12}}$ and in $\mathbb{F}_{2^6}$;
- Random generation of a polynomial in $\mathbb{F}_{2^{12}}$;
- Hash function "call".

Decapsulation

- Operations in $\mathbb{F}_{2^{12}}$ and in $\mathbb{F}_{2^6}$;
- Random generation of a polynomial in $\mathbb{F}_{2^{12}}$;
- Hash function "call".

DAGS Description

Key generation

The key generation process uses the following fundamental equation

$$\frac{1}{h_{i\oplus j}} = \frac{1}{h_i} + \frac{1}{h_j} + \frac{1}{h_0}. \tag{1}$$

To build the vector $\mathbf{h} = (h_0, \ldots, h_{n-1})$ of elements of $\mathbb{F}_{q^m}$, which is known as *signature* of a dyadic matrix.

DAGS Description

Key generation

1 Generate dyadic signature $\mathbf{h}$. To do this:

   i. Choose random non-zero distinct h_0 and h_j for $j = 2^l, l = 0, \ldots, \lfloor \log q^m \rfloor$.

   ii. Form the remaining elements using (1).

   iii. Return a selection of blocks of dimension s up to length n.

2 Build the Cauchy support. To do this:

   i. Choose a random offset $\omega \xleftarrow{\$} \mathbb{F}_{q^m}$.

   ii. Set $u_i = 1/h_i + \omega$ and $v_j = 1/h_j + 1/h_0 + \omega$ for $i = 0, \ldots, s-1$ and $j = 0, \ldots, n-1$.

   iii. Set $\mathbf{u} = (u_0, \ldots, u_{s-1})$ and $\mathbf{v} = (v_0, \ldots, v_{n-1})$.

DAGS Description

Key generation

3. Form Cauchy matrix $\hat{H}_1 = C(\mathbf{u}, \mathbf{v})$.
4. Build blocks $\hat{H}_i$, $i = 2, \ldots t$, by raising each element of $\hat{H}_1$ to the power of i.
5. Superimpose blocks to form matrix $\hat{H}$.
6. Choose random elements $z_i \xleftarrow{\$} \mathbb{F}_{q^m}$ such that $z_{is+j} = z_{is}$ for $i = 0, \ldots, n_0 - 1$, $j = 0, \ldots, s - 1$.
7. Form $H = \hat{H} \cdot \mathrm{Diag}(\mathbf{z})$.
8. Transform H into alternant form: call this H'.
9. Project H onto $\mathbb{F}_q$ using the co-trace function: call this H_{base}.
10. Write H_{base} in systematic form $(M \mid I_{n-k})$.
11. The public key is the generator matrix $G = (I_k \mid M^T)$.
12. The private key is the alternant matrix H'.

DAGS Description

Encapsulation

1. Choose $\mathbf{m} \xleftarrow{\$} \mathbb{F}_q^{k'}$.
2. Compute $\mathbf{r} = \mathcal{G}(\mathbf{m})$ and $\mathbf{d} = \mathcal{H}(\mathbf{m})$.
3. Parse $\mathbf{r}$ as $(\boldsymbol{\rho} \parallel \boldsymbol{\sigma})$ then set $\boldsymbol{\mu} = (\boldsymbol{\rho} \parallel \mathbf{m})$.
4. Generate error vector $\mathbf{e}$ of length n and weight w from $\boldsymbol{\sigma}$.
5. Compute $\mathbf{c} = \boldsymbol{\mu} G + \mathbf{e}$.
6. Compute $\mathbf{k} = \mathcal{K}(\mathbf{m})$.
7. Output ciphertext $(\mathbf{c}, \mathbf{d})$; the encapsulated key is $\mathbf{k}$.

DAGS Description

Decapsulation

1. Input private key, i.e. parity-check matrix H' in alternant form.
2. Use H' to decode $\mathbf{c}$ and obtain codeword $\boldsymbol{\mu}'G$ and error $\mathbf{e}'$.
3. Output $\perp$ if decoding fails or $(\mathbf{e}') \neq w$
4. Recover $\boldsymbol{\mu}'$ and parse it as $(\boldsymbol{\rho}' \parallel \mathbf{m}')$.
5. Compute $\mathbf{r}' = \mathcal{G}(\mathbf{m}')$ and $\mathbf{d}' = \mathcal{H}(\mathbf{m}')$.
6. Parse $\mathbf{r}'$ as $(\boldsymbol{\rho}'' \parallel \boldsymbol{\sigma}')$.
7. Generate error vector $\mathbf{e}''$ of length n and weight w from $\boldsymbol{\sigma}'$.
8. If $\mathbf{e}' \neq \mathbf{e}'' \vee \rho' \neq \rho'' \vee \mathbf{d} \neq \mathbf{d}'$ output $\perp$.
9. Else compute $\mathbf{k} = \mathcal{K}(\mathbf{m}')$.
10. The decapsulated key is $\mathbf{k}$.

Questions

Thank you for your attention.
Questions?

gustavo@cryptme.in
epersichetti@fau.edu