DAGS: Key Encapsulation using Dyadic GS Codes

Gustavo Banegas, 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

13 April 2018

FAU
Charles E. Schmidt College of Science
Florida Atlantic University

Classic Code-based Cryptography

Based on the hardness of decoding random linear codes (syndrome decoding problem).

Follows McEliece/Niederreiter framework.

Very efficient computation.

Natural implementation features thanks to binary vectors arithmetic.

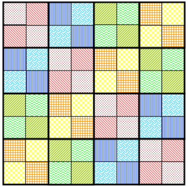
Drawback: large keys (around 1 MByte).

Try to tackle the large key issue.

Idea: public matrix with compact description.

Quasi-Cyclic Codes (as seen before).

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



Several code families have QD description:

If dyadic *signature* and code *support* verify certain conditions...

...then Dyadic $\cap$ Cauchy $\cap$ Goppa.

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 $\dfrac{z_j}{(v_j - u_\ell)^i}$, (distinct) nonzero elements of $\mathbb{F}_{q^m}$.

If $t = 1$ this is a Goppa code.

Can generate QD-GS codes using (modified) algorithm for QD Goppa (P. '12).

Efficient decoder, similar performance, more flexibility.

DAGS: A QD-GS BASED KEM

Select hash functions $\mathcal{G}, \mathcal{H}, \mathcal{K}$.

Select hash functions $\mathcal{G}, \mathcal{H}, \mathcal{K}$.

KEY GENERATION

- Generate a QD-GS code.
- SK: parity-check matrix H in alternant form over $\mathbb{F}_{q^m}$.
- PK: generator matrix G in systematic form over $\mathbb{F}_q$.

DAGS: A QD-GS BASED KEM

Select hash functions $\mathcal{G}, \mathcal{H}, \mathcal{K}$.

KEY GENERATION

- Generate a QD-GS code.
- SK: parity-check matrix H in alternant form over $\mathbb{F}_{q^m}$.
- PK: generator matrix G in systematic form over $\mathbb{F}_q$.

ENCAPSULATION

- Choose random word $\boldsymbol{m} \in \mathbb{F}_q^{k'}$.
- Compute $(\boldsymbol{\rho} \parallel \boldsymbol{\sigma}) = \mathcal{G}(\boldsymbol{m})$ and $\boldsymbol{d} = \mathcal{H}(\boldsymbol{m})$.
- Generate error vector $\boldsymbol{e} \in \mathbb{F}_q^n$ of weight w from seed $\boldsymbol{\sigma}$.
- Output $(\boldsymbol{c}, \boldsymbol{d})$ where $\boldsymbol{c} = (\boldsymbol{\rho} \parallel \boldsymbol{m})G + \boldsymbol{e}$ and set $\boldsymbol{k} = \mathcal{K}(\boldsymbol{m})$.

DAGS: a QD-GS based KEM

Select hash functions $\mathcal{G}, \mathcal{H}, \mathcal{K}$.

Key Generation

- Generate a QD-GS code.
- SK: parity-check matrix H in alternant form over $\mathbb{F}_{q^m}$.
- PK: generator matrix G in systematic form over $\mathbb{F}_q$.

Encapsulation

- Choose random word $\boldsymbol{m} \in \mathbb{F}_q^{k'}$.
- Compute $(\boldsymbol{\rho} \parallel \boldsymbol{\sigma}) = \mathcal{G}(\boldsymbol{m})$ and $\boldsymbol{d} = \mathcal{H}(\boldsymbol{m})$.
- Generate error vector $\boldsymbol{e} \in \mathbb{F}_q^n$ of weight w from seed $\boldsymbol{\sigma}$.
- Output $(\boldsymbol{c}, \boldsymbol{d})$ where $\boldsymbol{c} = (\boldsymbol{\rho} \parallel \boldsymbol{m})G + \boldsymbol{e}$ and set $\boldsymbol{k} = \mathcal{K}(\boldsymbol{m})$.

Decapsulation

- Recover codeword $((\boldsymbol{\rho}' \parallel \boldsymbol{m}')$ and error $\boldsymbol{e}')$ from $Decode(\boldsymbol{c})$.
- Recompute $\mathcal{G}(\boldsymbol{m}')$, $\mathcal{H}(\boldsymbol{m}')$ and $\boldsymbol{e}''$, then compare.
- Return $\perp$ if decoding fails or any check fails, else return $\boldsymbol{k} = \mathcal{K}(\boldsymbol{m}')$.

About DAGS

Uses McEliece framework and IND-CCA KEM transform (Hofheinz, Hövelmanns, Kiltz '17).

About DAGS

Uses McEliece framework and IND-CCA KEM transform (Hofheinz, Hövelmanns, Kiltz '17).

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

About DAGS

Uses McEliece framework and IND-CCA KEM transform (Hofheinz, Hövelmanns, Kiltz ’17).

Leverages “randomized” IND-CPA McEliece variant for tighter security proof.

Length k' of input $\boldsymbol{m}$ is kept short, but long enough for 256 bits of entropy.

About DAGS

Uses McEliece framework and IND-CCA KEM transform (Hofheinz, Hövelmanns, Kiltz ’17).

Leverages “randomized” IND-CPA McEliece variant for tighter security proof.

Length k' of input $\boldsymbol{m}$ is kept short, but long enough for 256 bits of entropy.

This helps keeping $\boldsymbol{d}$ small and making hashing more practical.

About DAGS

Uses McEliece framework and IND-CCA KEM transform (Hofheinz, Hövelmanns, Kiltz '17).

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

Length k' of input $\boldsymbol{m}$ is kept short, but long enough for 256 bits of entropy.

This helps keeping $\boldsymbol{d}$ small and making hashing more practical.

Private key (matrix H) can be efficiently represented by "support" $(\boldsymbol{v}, \boldsymbol{y})$.

About DAGS

Uses McEliece framework and IND-CCA KEM transform (Hofheinz, Hövelmanns, Kiltz '17).

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

Length k' of input $\boldsymbol{m}$ is kept short, but long enough for 256 bits of entropy.

This helps keeping $\boldsymbol{d}$ small and making hashing more practical.

Private key (matrix H) can be efficiently represented by "support" $(\boldsymbol{v}, \boldsymbol{y})$.

Alternant matrix is reconstructed on the fly together with syndrome computation.

About DAGS

Uses McEliece framework and IND-CCA KEM transform (Hofheinz, Hövelmanns, Kiltz '17).

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

Length k' of input $\boldsymbol{m}$ is kept short, but long enough for 256 bits of entropy.

This helps keeping $\boldsymbol{d}$ small and making hashing more practical.

Private key (matrix H) can be efficiently represented by "support" $(\boldsymbol{v}, \boldsymbol{y})$.

Alternant matrix is reconstructed on the fly together with syndrome computation.

This results in a small private key without computational overhead.

There exist structural attacks targeting structured alternant codes: FOPT and variants (Faugère, Otmani, Perret, Tillich ’10).

SECURITY

There exist structural attacks targeting structured alternant codes: FOPT and variants (Faugère, Otmani, Perret, Tillich '10).

QC/QD structure crucial to reduce number of unknowns of system.

Security

There exist structural attacks targeting structured alternant codes: FOPT and variants (Faugère, Otmani, Perret, Tillich '10).

QC/QD structure crucial to reduce number of unknowns of system.

No definitive complexity analysis available.

Security

There exist structural attacks targeting structured alternant codes: FOPT and variants (Faugère, Otmani, Perret, Tillich '10).

QC/QD structure crucial to reduce number of unknowns of system.

No definitive complexity analysis available.

Experimental evidence + (loose) theoretical bound

Security

There exist structural attacks targeting structured alternant codes: FOPT and variants (Faugère, Otmani, Perret, Tillich '10).

QC/QD structure crucial to reduce number of unknowns of system.

No definitive complexity analysis available.

Experimental evidence $+$ (loose) theoretical bound
$=$ hardness scales with dimension of solution space (number of free variables).

SECURITY

There exist structural attacks targeting structured alternant codes: FOPT and variants
(Faugère, Otmani, Perret, Tillich '10).

QC/QD structure crucial to reduce number of unknowns of system.

No definitive complexity analysis available.

Experimental evidence $+$ (loose) theoretical bound
$=$ hardness scales with dimension of solution space (number of free variables).

This is given by $m-1$ for QD Goppa, but it is $mt-1$ for QD-GS codes.

Security

There exist structural attacks targeting structured alternant codes: FOPT and variants (Faugère, Otmani, Perret, Tillich '10).

QC/QD structure crucial to reduce number of unknowns of system.

No definitive complexity analysis available.

Experimental evidence + (loose) theoretical bound
= hardness scales with dimension of solution space (number of free variables).

This is given by $m - 1$ for QD Goppa, but it is $mt - 1$ for QD-GS codes.

All QD Goppa parameters broken except for largest instances ($m = 16$).

SECURITY

There exist structural attacks targeting structured alternant codes: FOPT and variants (Faugère, Otmani, Perret, Tillich '10).

QC/QD structure crucial to reduce number of unknowns of system.

No definitive complexity analysis available.

Experimental evidence $+$ (loose) theoretical bound
$=$ hardness scales with dimension of solution space (number of free variables).

This is given by $m - 1$ for QD Goppa, but it is $mt - 1$ for QD-GS codes.

All QD Goppa parameters broken except for largest instances ($m = 16$).

No broken QD-GS parameters to date.

Performance

We choose:

- Small m
- Large s (power of 2)
- $t > 1$ odd
- Non-binary base field

PERFORMANCE

We choose:

- Small m
- Large s (power of 2)
- $t > 1$ odd
- Non-binary base field

$\mathbb{F}_{q^m} = \mathbb{F}_{2^N}$ large enough to define code, without being huge ($N \leq 12$).

Performance

We choose:

- Small m
- Large s (power of 2)
- $t > 1$ odd
- Non-binary base field

$\mathbb{F}_{q^m} = \mathbb{F}_{2^N}$ large enough to define code, without being huge ($N \leq 12$).

Stay clear of algebraic attacks ($mt > 21$).

Performance

We choose:

- Small m
- Large s (power of 2)
- $t > 1$ odd
- Non-binary base field

$\mathbb{F}_{q^m} = \mathbb{F}_{2^N}$ large enough to define code, without being huge ($N \leq 12$).

Stay clear of algebraic attacks ($mt > 21$).

High error-correction capacity ($st/2$) $\rightarrow$ smaller codes.

We choose:

- Small m
- Large s (power of 2)
- $t > 1$ odd
- Non-binary base field

$\mathbb{F}_{q^m} = \mathbb{F}_{2^N}$ large enough to define code, without being huge ($N \leq 12$).

Stay clear of algebraic attacks ($mt > 21$).

High error-correction capacity ($st/2$) $\rightarrow$ smaller codes.

Parameters (sizes in bytes):

q	m	n	k	k'	s	t	w	PK	SK	Ciphertext	Sec. Level
2^5	2	832	416	26	2^4	13	104	6,760	2,496	552	1
2^6	2	1216	512	43	2^5	11	176	8,448	3,648	944	3
2^6	2	2112	704	43	2^6	11	352	11,616	6,336	1,616	5

Ongoing and Future Work

Simple reference implementation, designed for clarity.

Ongoing and Future Work

Simple reference implementation, designed for clarity.

Implementation is isochronous, to resist timing attacks and the like.

Ongoing and Future Work

Simple reference implementation, designed for clarity.

Implementation is isochronous, to resist timing attacks and the like.

Much more efficient implementations are being developed:

- Vectorized/Assembly/C++
- Hardware (FPGA)
- ...

Ongoing and Future Work

Simple reference implementation, designed for clarity.

Implementation is isochronous, to resist timing attacks and the like.

Much more efficient implementations are being developed:

- Vectorized/Assembly/C++
- Hardware (FPGA)
- ...

Several optimizations from practice/theory are being investigated.

Ongoing and Future Work

Simple reference implementation, designed for clarity.

Implementation is isochronous, to resist timing attacks and the like.

Much more efficient implementations are being developed:

- Vectorized/Assembly/C++
- Hardware (FPGA)
- ...

Several optimizations from practice/theory are being investigated.

Work in progress to make implementation side-channel resistant.

Ongoing and Future Work

Simple reference implementation, designed for clarity.

Implementation is isochronous, to resist timing attacks and the like.

Much more efficient implementations are being developed:

- Vectorized/Assembly/C++
- Hardware (FPGA)
- ...

Several optimizations from practice/theory are being investigated.

Work in progress to make implementation side-channel resistant.

Accurate complexity analysis of algebraic attacks is ongoing/future project.

Conclusions

DAGS has many good features:

Conclusions

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less

CONCLUSIONS

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability

Conclusions

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for “binary DAGS” is being developed

CONCLUSIONS

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for "binary DAGS" is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys

Conclusions

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for “binary DAGS” is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys
- Efficient in practice

CONCLUSIONS

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for “binary DAGS” is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys
- Efficient in practice
  - Preliminary results in hardware show a speedup of up to 46x

Conclusions

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for "binary DAGS" is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys
- Efficient in practice
  - Preliminary results in hardware show a speedup of up to 46x
  - e.g. timing of 78,318 ns for DAGS_3 Encapsulation

Conclusions

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for “binary DAGS” is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys
- Efficient in practice
  - Preliminary results in hardware show a speedup of up to 46x
  - e.g. timing of 78,318 ns for DAGS_3 Encapsulation
- Entirely patent-free

Conclusions

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for “binary DAGS” is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys
- Efficient in practice
  - Preliminary results in hardware show a speedup of up to 46x
  - e.g. timing of 78,318 ns for DAGS_3 Encapsulation
- Entirely patent-free

Some delicate points:

Conclusions

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for "binary DAGS" is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys
- Efficient in practice
  - Preliminary results in hardware show a speedup of up to 46x
  - e.g. timing of 78,318 ns for DAGS_3 Encapsulation
- Entirely patent-free

Some delicate points:

- Caution required with structural attacks

Conclusions

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for “binary DAGS” is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys
- Efficient in practice
  - Preliminary results in hardware show a speedup of up to 46x
  - e.g. timing of 78,318 ns for DAGS_3 Encapsulation
- Entirely patent-free

Some delicate points:

- Caution required with structural attacks
  - Easy to avoid with appropriate choice of parameters

Conclusions

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
  - Option for "binary DAGS" is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys
- Efficient in practice
  - Preliminary results in hardware show a speedup of up to 46x
  - e.g. timing of 78,318 ns for DAGS_3 Encapsulation
- Entirely patent-free

Some delicate points:

- Caution required with structural attacks
  - Easy to avoid with appropriate choice of parameters
  - Folding attacks don't perform well on large (non-binary) base field

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters $\rightarrow$ high flexibility and scalability
    - Option for "binary DAGS" is being developed
- Alternant decoding presents no decryption failures $\rightarrow$ allows use of static keys
- Efficient in practice
    - Preliminary results in hardware show a speedup of up to 46x
    - e.g. timing of 78,318 ns for DAGS_3 Encapsulation
- Entirely patent-free

Some delicate points:

- Caution required with structural attacks
    - Easy to avoid with appropriate choice of parameters
    - Folding attacks don't perform well on large (non-binary) base field
- Non-binary arithmetic $\rightarrow$ more complex implementation

CONCLUSIONS

DAGS has many good features:

- Small sizes for all data (pk, sk, ciphertext): few Kb or less
- Many intertwined parameters → high flexibility and scalability
  - Option for “binary DAGS” is being developed
- Alternant decoding presents no decryption failures → allows use of static keys
- Efficient in practice
  - Preliminary results in hardware show a speedup of up to 46x
  - e.g. timing of 78,318 ns for DAGS_3 Encapsulation
- Entirely patent-free

Some delicate points:

- Caution required with structural attacks
  - Easy to avoid with appropriate choice of parameters
  - Folding attacks don’t perform well on large (non-binary) base field
- Non-binary arithmetic → more complex implementation
  - Price to pay is actually fairly small

Thank you

$\bowtie$

www.dags-project.org