Code-based Signatures from Secure Multiparty Computation

Thibauld Feneuil

SIAM AG23 — Advances in Code-based Signature

July 12, 2023, Eindhoven

Registration and travel support for this presentation was provided by the Eindhoven University of Technology.
Introduction
Proof of knowledge

I know $x$ such that $F(x) = y$.

- **Completeness:** $\Pr[\text{verif } \checkmark \mid \text{honest prover}] = 1$
- **Soundness:** $\Pr[\text{verif } \checkmark \mid \text{malicious prover}] \leq \epsilon$ (e.g. $2^{-128}$)
- **Zero-knowledge:** verifier learns nothing on $x$
MPC in the Head

- [IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: “Zero-knowledge from secure multiparty computation” (STOC 2007)
- Turn an MPC protocol into a zero knowledge proof of knowledge
- **Generic**: can be apply to any cryptographic problem
- Convenient to build (candidate) **post-quantum signature** schemes
- **Picnic**: submission to NIST (2017)
- Recent NIST call (01/06/2023): 7 MPCitH schemes / 50 submissions
One-way function

\[ F : x \mapsto y \]

E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Input sharing \([x]\)

Joint evaluation of:

\[ g(x) = \begin{cases} 
  \text{Accept} & \text{if } F(x) = y \\
  \text{Reject} & \text{if } F(x) \neq y 
\end{cases} \]

Signature scheme

\[ m \]

Signature

Hash function

Zero-knowledge proof

Prover

\[ x \]

Verifier

\[ y \]

OK you know \( x \)
One-way function

\[ F : x \mapsto y \]

E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Input sharing \([x]\)

Joint evaluation of:

\[ g(x) = \begin{cases} 
\text{Accept} & \text{if } F(x) = y \\
\text{Reject} & \text{if } F(x) \neq y 
\end{cases} \]

Signature scheme

\[ x \xrightarrow{\text{msg}} \text{Hash function} \xrightarrow{\text{signature}} \]

Zero-knowledge proof

Prover \(x\) \xrightarrow{\text{OK you know } x} \text{Verifier } y
One-way function

\[ F : x \mapsto y \]
E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Input sharing \( [[x]] \)
Joint evaluation of:

\[ g(x) = \begin{cases} 
\text{Accept} & \text{if } F(x) = y \\
\text{Reject} & \text{if } F(x) \neq y 
\end{cases} \]

Signature scheme

\( x \)

Hash function

signature

Zero-knowledge proof

\( x \)

Prover

\( y \)

Verifier

OK you know \( x \)

\[ [[x]] = ([x]_1, \ldots, [x]_N) \quad \text{s.t.} \quad x = [x]_1 + \ldots + [x]_N \]
One-way function

\[ F : x \mapsto y \]

E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Input sharing \([x]\)

Joint evaluation of:

\[ g(x) = \begin{cases} 
\text{Accept} & \text{if } F(x) = y \\
\text{Reject} & \text{if } F(x) \neq y 
\end{cases} \]

Signature scheme

\( x \)

Hash function

signature

Zero-knowledge proof

Prover

\( x \)

Verifier

\( y \)

OK you know \( x \)
One-way function

\[ F : x \mapsto y \]

E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Input sharing \([x]\)

Joint evaluation of:

\[ g(x) = \begin{cases} 
\text{Accept} & \text{if } F(x) = y \\
\text{Reject} & \text{if } F(x) \neq y
\end{cases} \]

Zero-knowledge proof

Prover

Verifier

OK you know \(x\)
One-way function

\[ F : x \mapsto y \]

E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Input sharing \( [[x]] \)
Joint evaluation of:

\[ g(x) = \begin{cases} 
\text{Accept} & \text{if } F(x) = y \\
\text{Reject} & \text{if } F(x) \neq y 
\end{cases} \]

Zero-knowledge proof

MPC in the Head transform

Signature scheme

Hash function

\( \text{signature} \)

\( \text{msg} \)
MPCitH: general principle
MPC model

- Jointly compute
  \[ g(x) = \begin{cases} 
  \text{Accept} & \text{if } F(x) = y \\
  \text{Reject} & \text{if } F(x) \neq y 
  \end{cases} \]

- \((N - 1)\) private: the views of any \(N - 1\) parties provide no information on \(x\)

- Semi-honest model: assuming that the parties follow the steps of the protocol
- **Jointly compute**

\[ g(x) = \begin{cases} 
\text{Accept} & \text{if } F(x) = y \\
\text{Reject} & \text{if } F(x) \neq y 
\end{cases} \]

- **\((N - 1)\) private**: the views of any \(N - 1\) parties provide no information on \(x\)

- **Semi-honest model**: assuming that the parties follow the steps of the protocol

- **Broadcast model**
  - Parties locally compute on their shares \([x] \mapsto [\alpha]\)
  - Parties broadcast \([\alpha]\) and recompute \(\alpha\)
  - Parties start again (now knowing \(\alpha\))
MPCitH transform

Prover

Verifier
MPCitH transform

1. Generate and commit shares
   \([x] = ([x]_1, \ldots, [x]_N)\)

Prover

Verifier

\[ \text{Com}^\rho_1([x]_1) \]
\[ \ldots \]
\[ \text{Com}^\rho_N([x]_N) \]
MPCitH transform

① Generate and commit shares
\[ [x] = ([x]_1, \ldots, [x]_N) \]

② Run MPC in their head

Prover

Verifier
MPCitH transform

1. Generate and commit shares
   \[
   [[x]] = ([[[x]]_1, \ldots, [[[x]]_N])
   \]

2. Run MPC in their head

   ![Diagram of MPC in their head]

   Prover

   Com^{\rho_1}([[x]]_1)
   \[\cdots\]
   Com^{\rho_N}([[x]]_N)
   
   send broadcast
   \[
   [[[\alpha]]_1, \ldots, [[[\alpha]]_N]
   \]

3. Choose a random party
   \[i^* \leftarrow \{1, \ldots, N\}\]

Verifier
MPCitH transform

1. Generate and commit shares
\[ [[x]] = ([[x]]_1, ..., [[x]]_N) \]

2. Run MPC in their head

3. Choose a random party
\[ i^* \leftarrow \{1, \ldots, N\} \]

4. Open parties \( \{1, \ldots, N\} \setminus \{i^*\} \)

Prover

Verifier
① Generate and commit shares
\[ [x] = ([x]_1, \ldots, [x]_N) \]

② Run MPC in their head

③ Choose a random party
\[ i^* \leftarrow \{1, \ldots, N\} \]

④ Open parties \( \{1, \ldots, N\} \backslash \{i^*\} \)

⑤ Check \( \forall i \neq i^* \)
- Commitments Com\(^{\rho_i}([x]_i)\)
- MPC computation \([\alpha]_i = \varphi([x]_i)\)
Check \( g(y, \alpha) = \text{Accept} \)
① Generate and commit shares
\[
[x] = ([x]_1, \ldots, [x]_N)
\]
We have \( F(x) \neq y \) where
\[
x := [x]_1 + \ldots + [x]_N
\]

\[
\text{MPCitH transform}
\]

\[
\begin{align*}
\text{Com}^{\rho_1}([x]_1) \\
\vdots \\
\text{Com}^{\rho_N}([x]_N)
\end{align*}
\]
MPCitH transform

1. Generate and commit shares
   \[ [x] = ([x]_1, \ldots, [x]_N) \]
   
   We have \( F(x) \neq y \) where
   \[ x := [x]_1 + \ldots + [x]_N \]

2. Run MPC in their head

\[ \text{Com}^\mu([x]_1) \]
\[ \ldots \]
\[ \text{Com}^\nu([x]_N) \]

send broadcast
\[ \|\alpha\|_1, \ldots, \|\alpha\|_N \]
MPCitH transform

1. Generate and commit shares
   \[ [[x]] = ([[x]]_1, \ldots, [[x]]_N) \]
   We have \( F(x) \neq y \) where
   \( x := [[x]]_1 + \ldots + [[x]]_N \)

2. Run MPC in their head

3. Choose a random party
   \( i^* \leftarrow \mathcal{S} \{1, \ldots, N\} \)

Malicious Prover

Verifier
MPCitH transform

① Generate and commit shares
\[
[[x]] = ([[[x]_1], \ldots, [[[x]]_N])
\]
We have \( F(x) \neq y \) where
\[
x := [[[x]]_1 + \ldots + [[[x]]_N]
\]

② Run MPC in their head

③ Choose a random party
\( i^* \leftarrow \{1, \ldots, N\} \)

④ Open parties \( \{1, \ldots, N\} \setminus \{i^*\} \)

Malicious Prover

Verifier
**MPCitH transform**

1. Generate and commit shares
   \[ \boxed{[x]} = ([x]_1, \ldots, [x]_N) \]

   We have \( F(x) \neq y \) where
   \[ x := [x]_1 + \ldots + [x]_N \]

2. Run MPC in their head

3. Choose a random party
   \[ i^* \leftarrow^$ \{1, \ldots, N\} \]

4. Open parties \( \{1, \ldots, N\} \setminus \{i^*\} \)

5. Check \( \forall i \neq i^* \)
   - Commitments \( \text{Com}^\rho([x]_i) \)
   - MPC computation \( [\alpha]_i = \varphi([x]_i) \)
   - Check \( g(y, \alpha) = \text{Accept} \)

---

**Malicious Prover**

**Verifier**

*Cheating detected!*
MPCitH transform

1. Generate and commit shares
   \([x] = ([x]_1, \ldots, [x]_N)\)
   \(\text{We have } F(x) \neq y\text{ where } x := [x]_1 + \ldots + [x]_N\)

2. Run MPC in their head

3. Choose a random party
   \(i^* \leftarrow \{1, \ldots, N\}\)

4. Open parties \(\{1, \ldots, N\} \setminus \{i^*\}\)

5. Check \(\forall i \neq i^*\)
   - Commitments \(\text{Com}^\rho([x]_i)\)
   - MPC computation \([\alpha]_i = \varphi([x]_i)\)
   - Check \(g(y, \alpha) = \text{Accept}\)

Malicious Prover

Verifier

\(\checkmark\) Seems OK.
MPCitH transform

- Zero-knowledge \iff\ MPC protocol is \((N - 1)\)-private
MPCitH transform

- Zero-knowledge $\iff$ MPC protocol is $(N - 1)$-private

- Soundness:

\[ \Pr(\text{malicious prover convinces the verifier}) = \Pr(\text{corrupted party remains hidden}) = \frac{1}{N} \]
MPCitH transform

- **Zero-knowledge** ⇔ MPC protocol is \((N - 1)\)-private

- **Soundness:**

  \[
  \mathbb{P}(\text{malicious prover convinces the verifier}) = \mathbb{P}(\text{corrupted party remains hidden}) = \frac{1}{N}
  \]

- **Parallel repetition**

  Protocol repeated \(\tau\) times in parallel → soundness error \(\left(\frac{1}{N}\right)^\tau\)
Code-based signature schemes
One-way function

\[ F : x \mapsto y \]

E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Joint evaluation of:

\[ g(x) = \begin{cases} 
\text{Accept} & \text{if } F(x) = y \\
\text{Reject} & \text{if } F(x) \neq y 
\end{cases} \]

Signature scheme

\( x \xrightarrow{\text{message}} \text{Hash function} \xrightarrow{\text{signature}} \)

Zero-knowledge proof

\( x \xrightarrow{\text{Prover}} \text{Verifier} \xrightarrow{\text{OK you know } x} \)
One-way function

\[ F : x \mapsto y \]

E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Input sharing \([x]\)

Joint evaluation of:

\[ g(x) = \begin{cases} 
  \text{Accept} & \text{if } F(x) = y \\
  \text{Reject} & \text{if } F(x) \neq y
\end{cases} \]

Zero-knowledge proof

MPC in the Head transform

Signature scheme

Hash function

Signature scheme

\[ x \]

\[ \text{signature} \]

\[ \text{msg} \]

\[ \text{Hash function} \]

Prover

Verifier

OK you know \(x\)
One-way function

\[ F : x \mapsto y \]

E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Input sharing \[[x]\]

Joint evaluation of:

\[ g(x) = \begin{cases} 
\text{Accept} & \text{if } F(x) = y \\
\text{Reject} & \text{if } F(x) \neq y
\end{cases} \]

Signature scheme

Prover

Verifier

Zero-knowledge proof

\[ x \rightarrow \text{signature} \]

\[ x \rightarrow \text{Hash function} \]

\[ x \rightarrow y \]

OK you know \( x \)

Fiat-Shamir transform
One-way function

\[ F : x \mapsto y \]

E.g. AES, MQ system, Syndrome decoding

Multiparty computation (MPC)

Input sharing \( \llbracket x \rrbracket \)
Joint evaluation of:

\[ g(x) = \begin{cases} 
\text{Accept} & \text{if } F(x) = y \\
\text{Reject} & \text{if } F(x) \neq y 
\end{cases} \]

Signature scheme

Prover

Zero-knowledge proof

Verifier

OK you know \( x \)
Submitted code-based candidates at NIST call

<table>
<thead>
<tr>
<th>Syndrome Decoding Problem in Hamming metric</th>
<th>Syndrome Decoding Problem in rank metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given a matrix $H$ and a vector $y$, find $x$ such that $y = Hx$ and $\text{wt}_H(x) \leq w$</td>
<td>$\text{wt}<em>R(x) \leq r$ \hspace{1cm} ($x \in \mathbb{F}</em>{q^m}^n$)</td>
</tr>
</tbody>
</table>
Submitted code-based candidates at NIST call

<table>
<thead>
<tr>
<th>Syndrome Decoding Problem in Hamming metric</th>
<th>Syndrome Decoding Problem in rank metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given a matrix $H$ and a vector $y$, find $x$ such that $y = Hx$ and $\text{wt}_H(x) \leq w$</td>
<td>$\text{wt}<em>R(x) \leq r$ (for $x \in \mathbb{F}</em>{q^m}^n$)</td>
</tr>
<tr>
<td>The MPC protocol checks that the shared input $x$ satisfies $y = Hx$ and there exists a degree-$w$ polynomial $Q$ such that</td>
<td></td>
</tr>
<tr>
<td>$\forall i, x_i \cdot Q(y_i) = 0.$</td>
<td>$\forall i, L(x_i) = 0.$</td>
</tr>
</tbody>
</table>


[Fen22] Feneuil T. Building MPCitH-based Signatures from MQ, MinRank, Rank SD and PKP. ePrint 2022-1512
Submitted code-based candidates at NIST call

Syndrome Decoding Problem in *Hamming* metric

Given a matrix $H$ and a vector $y$, find $x$ such that $y = Hx$ and

$$\text{wt}_H(x) \leq w$$

The MPC protocol checks that the shared input $x$ satisfies $y = Hx$ and there exists a degree-$w$ polynomial $Q$ such that

$$\forall i, \ x_i \cdot Q(\gamma_i) = 0.$$  

**SD-in-the-Head (SDitH)**


Syndrome Decoding Problem in rank metric

There exists a degree-$q^r$ $q$-polynomial $L := \sum_{i=0}^{r} L_i X^{q^i}$ such that

$$\forall i, L(x_i) = 0.$$  

**RYDE**

### Performances

<table>
<thead>
<tr>
<th></th>
<th>Short Instance</th>
<th>Fast Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>sig</td>
</tr>
<tr>
<td>SDitH-256</td>
<td>8.3</td>
<td>13.4</td>
</tr>
<tr>
<td>SDitH-251</td>
<td>8.3</td>
<td>22.1</td>
</tr>
<tr>
<td>RYDE</td>
<td>6.0</td>
<td>23.4</td>
</tr>
</tbody>
</table>

*Additive sharing*

*Shamir’s sharing*

*NIST Category I*

*Isochronous implementations*

*Size in kilobytes, timing in Mcycles*

@2.60GHz: 1 millisecond $\approx$ 2.6 Mcycles
**Performances**

How it scales for higher security levels?

<table>
<thead>
<tr>
<th>Category</th>
<th>SDitH</th>
<th>RYDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>pk</td>
</tr>
<tr>
<td>Category I</td>
<td>120 B</td>
<td>8.3 KB</td>
</tr>
<tr>
<td>Category III</td>
<td>183 B</td>
<td>19.2 KB</td>
</tr>
<tr>
<td>Category V</td>
<td>234 B</td>
<td>33.4 KB</td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>lpkI</th>
<th>lsigI</th>
<th>lsigI+lpki</th>
<th>t_sign</th>
<th>t_verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDitH-256-short</td>
<td>120</td>
<td>8241</td>
<td>8361</td>
<td>13.4</td>
<td>12.5</td>
</tr>
<tr>
<td>SDitH-251-short</td>
<td>120</td>
<td>8241</td>
<td>8361</td>
<td>22.1</td>
<td>21.2</td>
</tr>
<tr>
<td>SDitH-256-fast</td>
<td>120</td>
<td>10117</td>
<td>10237</td>
<td>5.1</td>
<td>1.6</td>
</tr>
<tr>
<td>SDitH-251-fast</td>
<td>120</td>
<td>10117</td>
<td>10237</td>
<td>4.4</td>
<td>0.6</td>
</tr>
<tr>
<td>RYDE-short</td>
<td>86</td>
<td>5956</td>
<td>6042</td>
<td>23.4</td>
<td>20.1</td>
</tr>
<tr>
<td>RYDE-fast</td>
<td>86</td>
<td>7446</td>
<td>7532</td>
<td>5.4</td>
<td>4.4</td>
</tr>
<tr>
<td>PERK-short3</td>
<td>150</td>
<td>6560</td>
<td>6710</td>
<td>39</td>
<td>27</td>
</tr>
<tr>
<td>PERK-short5</td>
<td>240</td>
<td>6060</td>
<td>6300</td>
<td>36</td>
<td>25</td>
</tr>
<tr>
<td>PERK-fast3</td>
<td>150</td>
<td>8350</td>
<td>8500</td>
<td>7.6</td>
<td>5.3</td>
</tr>
<tr>
<td>PERK-fast5</td>
<td>240</td>
<td>8030</td>
<td>8270</td>
<td>7.2</td>
<td>5.1</td>
</tr>
<tr>
<td>CROSS-fast</td>
<td>61</td>
<td>12944</td>
<td>13005</td>
<td>6.8</td>
<td>3.2</td>
</tr>
<tr>
<td>CROSS-small</td>
<td>61</td>
<td>10304</td>
<td>10365</td>
<td>22.0</td>
<td>10.3</td>
</tr>
<tr>
<td>CROSS-G-fast</td>
<td>32</td>
<td>8665</td>
<td>8697</td>
<td>11.0</td>
<td>7.8</td>
</tr>
<tr>
<td>CROSS-G-small</td>
<td>32</td>
<td>7625</td>
<td>7657</td>
<td>26.6</td>
<td>21.3</td>
</tr>
<tr>
<td>LESS-1b</td>
<td>13700</td>
<td>8400</td>
<td>22100</td>
<td>263.6</td>
<td>271.4</td>
</tr>
<tr>
<td>LESS-1i</td>
<td>41100</td>
<td>6100</td>
<td>47200</td>
<td>254.3</td>
<td>263.4</td>
</tr>
<tr>
<td>LESS-1s</td>
<td>95900</td>
<td>5200</td>
<td>101100</td>
<td>206.6</td>
<td>213.4</td>
</tr>
<tr>
<td>MEDS-9923</td>
<td>9923</td>
<td>9896</td>
<td>19819</td>
<td>518.1</td>
<td>515.6</td>
</tr>
<tr>
<td>MEDS-13220</td>
<td>13220</td>
<td>12976</td>
<td>26196</td>
<td>88.9</td>
<td>46.0</td>
</tr>
<tr>
<td>FuLeeca</td>
<td>1318</td>
<td>1100</td>
<td>2418</td>
<td>1846.8</td>
<td>1.26</td>
</tr>
<tr>
<td>Wave-822</td>
<td>3677390</td>
<td>822</td>
<td>3678212</td>
<td>1160</td>
<td>1.23</td>
</tr>
</tbody>
</table>

https://wave-sign.org/
https://www.meds-pqc.org/
https://www.less-project.com/
https://www.cross-crypto.com/
https://pqc-perk.org/
https://pqc-ryde.org/
https://sdith.org/
https://www.ce.cit.tum.de/.../fuleeca/
Comparison

The figure shows a scatter plot comparing the signature size (in bytes) against the public key size (in bytes) for various cryptographic schemes. The x-axis represents the public key size, while the y-axis represents the signature size. Different symbols and colors correspond to different cryptographic schemes, such as SPHINCS+, Dilithium, Falcon, SDitH, RYDE, CROSS, PERK, LESS, MEDS, FuLeeca, and Wave. The plot visually demonstrates the relationship between the two metrics for these schemes.
Comparison

![Comparison Graph]

- Signature Size (in bytes)
- Public Key Size (in bytes)
- Various algorithms represented with different markers:
  - SPHINCS+ (triangle)
  - Dilithium (red triangle)
  - Falcon (blue triangle)
  - SDiH
  - RYDE
  - CROSS
  - PERK
  - LESS
  - MDES
  - FuLeeca
  - Wave

The graph illustrates the comparison between different cryptographic algorithms based on their signature and public key sizes.
Comparison

[Graph showing the comparison of signature and public key sizes for various cryptographic schemes. The graph plots the signature size against the public key size on a logarithmic scale.

- LESS is marked with a symbol indicating it has a smaller signature size compared to other schemes.
- MEDS is marked with a symbol indicating it has a larger public key size.

Legend:
- SPHINCS+
- Dilithium
- Falcon
- SDitH
- RYDE
- CROSS
- PERK
- LESS
- MEDS
- FuLeeca
- Wave]
Comparison
Comparison

- Signature Size (in bytes)

- Public Key Size (in bytes)

- Different cryptographic schemes represented with various markers.
Comparison

SDitH, RYDE (MPCitH)
Comparison

The diagram compares the signature size and public key size of various cryptographic schemes. The x-axis represents the public key size in bytes, while the y-axis represents the signature size in bytes. Different symbols and colors are used to represent different schemes, such as SPHINCS+, Dilithium, Falcon, SDitH, RYDE, CROSS, PERK, LESS, MEDS, FuLeecia, and Wave.
Comparison

Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.
Comparison

Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.
Comparison

Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.
Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.
Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.
Optimized implementations are not available for all these signature schemes. These numbers will change in the coming months.
Advantages and limitations

- **Limitations**
  - Relatively *slow* (few milliseconds)
    - Greedy use of symmetric cryptography
  - Relatively *large* signatures (5-10 KB for L1)
  - *Quadratic* growth in the security level

- **Advantages**
  - *Conservative* hardness assumption:
    - Old problems, no structure, no trapdoor
  - *Small* (public) keys
  - Highly *parallelizable*
  - *Good* public key + signature size
  - Adaptive and *tunable* parameters
Conclusion

MPC-in-the-Head

- Very versatile and tunable
- Can be applied on any one-way function
- A practical tool to build conservative signature schemes
  - *No structure* in the security assumption
Conclusion

■ **MPC-in-the-Head**
  ■ Very versatile and tunable
  ■ Can be applied on any one-way function
  ■ A practical tool to build *conservative* signature schemes
    ■ *No structure* in the security assumption

■ **Perspectives**
  ■ *Additive-based MPCitH*: stable
  ■ *Low-threshold-based MPCitH*: new approach, could lead to follow-up works
Conclusion

- **MPC-in-the-Head**
  - Very versatile and tunable
  - Can be applied on any one-way function
  - A practical tool to build conservative signature schemes
    - *No structure* in the security assumption

- **Perspectives**
  - *Additive-based MPCitH*: stable
  - *Low-threshold-based MPCitH*: new approach, could lead to follow-up works

- **Remark:**
  - Can be applied to one-way functions from other research fields: multivariate quadratic problem, MinRank problem, …
Conclusion

SD-in-the-Head (SDitH)
C. Aguilar Melchor, T. Feneuil, N. Gama, S. Gueron,
J. Howe, D. Joseph, A. Joux, E. Persichetti,
T. Randrianarisoa, M. Rivain, D. Yue.

Website: https://sdith.org/
Email: consortium@sdith.org

RYDE
N. Aragon, M. Bardet, L. Bidoux, J.-J. Chi-Domínguez,
V. Dyseryn, T. Feneuil, P. Gaborit, A. Joux,

Website: https://pqc-ryde.org/
Email: team@pqc-ryde.org