The Return of the SDiTH

C. Aguilar-Melchor, N. Gama, J. Howe, A. Hülsing, D. Joseph, and S. Yue

Eurocrypt 2023 - April 2023
1. Hypercube MPC-in-the-Head:
   How to make MPC-in-the-Head faster keeping the same proof size.

2. Hypercube SDitH:
   A smaller post-quantum signature based on Syndrome Decoding in the Head.
Hypercube MPC-in-the-Head

Making digital signatures smaller and more secure

MPC-in-the-head + Fiat-Shamir

- **Hard instance**: Pick an instance of your favorite hard NP problem.
- **Fast MPC**: Evaluate its verification function in MPC
- **MPC-in-the-head**: Turns it into a zero knowledge proof of knowledge – malicious prover
- **Fiat-Shamir**: make it non interactive and turns it into a strong digital signature
  - Security is the one of solving the hard NP problem.
  - Signing oracle access does not bring any advantage.
Hypercube MPC-in-the-Head

Making digital signatures smaller and more secure

MPC-in-the-head + Fiat-Shamir

- **Hard instance**: Pick an instance of your favorite hard NP problem.
- **fast MPC**: Evaluate its verification function in MPC
- **MPC-in-the-head**: Turns it into a zero knowledge proof of knowledge – malicious prover
- **Fiat-Shamir**: make it non interactive and turns it in a strong digital signature
  - Security is the one of solving the hard NP problem.
  - Signing oracle access does not bring any advantage.
Picking an MPC framework

- Any number of players, the more, the better!
- Prefer linear/additive secret sharing protocol with public broadcasts.
- Target semi-honest security at this step
  *malicious security is regained later*
- Even a Trusted Dealer setup is ok!
  *provide any triplets as part of the inputs, and make sure the algorithm checks the triplet consistency.*

\[ \rightarrow \text{MPCitH operates in the fastest and most concise out of all MPC settings} \]

MPC algorithm: coding guidelines

- Optimize: \(|\text{inputs}|\) and \(|\text{communications}|\), bonus: running time and rounds.
Choice of MPC framework and algorithms

Picking an MPC framework

- Any number of players, the more, the better!
- Prefer linear/additive secret sharing protocol with public broadcasts.
- Target semi-honest security at this step
  
  *malicious security is regained later*

- Even a Trusted Dealer setup is ok!

  *provide any triplets as part of the inputs, and make sure the algorithm checks the triplet consistency.*

⇒ **MPCitH operates in the fastest and most concise out of all MPC settings**

MPC algorithm: coding guidelines

- Optimize: |inputs| and |communications|, bonus: running time and rounds.
Choice of MPC framework and algorithms

Picking an MPC framework

- Any number of players, the more, the better!
- Prefer linear/additive secret sharing protocol with public broadcasts.
- Target semi-honest security at this step
  *malicious security is regained later*
- Even a Trusted Dealer setup is ok!
  *provide any triplets as part of the inputs, and make sure the algorithm checks the triplet consistency.*

→ MPCitH operates in the fastest and most concise out of all MPC settings

MPC algorithm: coding guidelines

- Optimize: |inputs| and |communications|, bonus: running time and rounds.
How MPC-in-the-Head works - Full Threshold security

Prover - Simulates the MPC protocol in the head
- Commits to everything that is secret (i.e. input secret-shares)
- Publishes everything that is public (i.e. broadcasted communications).

Verifier - checks the result and detects cheats
- Asks the prover to open $N - 1$ parties inputs.
- Re-evaluate those parties and verify they have not cheated.

Bottom line: HVZK proof
- The verifier does not learn anything except the result.
- A prover that commits to secret shares that do not pass the verification function, gets caught with proba $1 - \frac{1}{N}$.
Complexity of MPC-in-the-Head

Player 1
Input share
\(x_1 \ u_1 \ d_1\)

Player \(n\)
Input share
\(x_n \ u_n \ d_n\)

\[f(\text{input}) = \alpha_1 \ \alpha_2 \ \ldots \ \alpha_n\]
\[g_{\alpha}(\text{input}) = \beta_1 \ \beta_2 \ \ldots \ \beta_n\]
\[z_{\alpha,\beta}(\text{input}) = \text{result}_1 \ \text{result}_2 \ \ldots \ \text{result}_n\]

broadcasts bulletin board
(revealed shares)

inputs hash
Plain texts:
(revealed values or masked values)

\(\text{commit}_1\)

\(\text{commit}_n\)

\(\text{coms hash}\)

\(\text{result} = 0\)
Complexity of MPC-in-the-Head

Player 1

Input share
\[ x_1 \ u_1 \ d_1 \]

commit_1

Player \( n \)

Input share
\[ x_n \ u_n \ d_n \]

commit_\( n \)

plain Input
\[ x \ \text{unif dep.} \]

Broadcasts Bulletin Board
(revealed shares)

\[ f(\text{input}) \]
\[ g_\alpha(\text{input}) \]
\[ z_{\alpha,\beta}(\text{input}) \]

(result_1 \ result_2 \ \ldots \ result_n)

\[ \alpha_1 \ \alpha_2 \ \ldots \ \alpha_n \]
\[ \beta_1 \ \beta_2 \ \ldots \ \beta_n \]

\( \text{inputs hash} \)

Plaintexts:
(revealed values or masked values)

\[ \alpha \]
\[ \beta \]

\( \text{result} = 0 \)

\( \text{coms hash} \)

CLEAR RUNNING TIME

BOTTLENECK

(PROOF AND VERIFICATION)
Complexity of MPC-in-the-Head

- **Before:** \( n \) evaluations of the MPC protocol (bottleneck)
- **Hypercube-MPCitH:** \( \log_2(n) \) evaluations of the MPC protocol (negligible)

**Main idea**
- **Before:** we evaluate each individual party
- **Hypercube-MPCitH:**
  - We group parties together and evaluate only \( \log_2(n) \) subsets of parties.
  - Groups of parties are defined geometrically by their coordinates on a Hypercube.
Partitioning the parties - Sub-MPC protocols

Original 6-players Protocol (chances of cheating: 1/6):

- Party 1: $x_1$
- Party 2: $x_2$
- Party 3: $x_3$
- Party 4: $x_4$
- Party 5: $x_5$
- Party 6: $x_6$

bcast: $\alpha_1, \beta_1, \ldots, \text{result}_1$

bcast: $\alpha_2, \beta_2, \ldots, \text{result}_2$

bcast: $\alpha_3, \beta_3, \ldots, \text{result}_3$

bcast: $\alpha_4, \beta_4, \ldots, \text{result}_4$

bcast: $\alpha_5, \beta_5, \ldots, \text{result}_5$

bcast: $\alpha_6, \beta_6, \ldots, \text{result}_6$
Partitioning the parties - Sub-MPC protocols

\[
\begin{array}{ccc}
\text{Party 1} & \text{Party 2} & \text{Party 3} \\
x_1 & x_2 & x_3 \\
\text{Party 4} & \text{Party 5} & \text{Party 6} \\
x_4 & x_5 & x_6 \\
\end{array}
\]

Plaintext Protocol:
- Plaintext: \( x_1 + \cdots + x_6 \)
- plain bcasts: \( \alpha, \beta, \ldots, \)result

Original 6-players Protocol (chances of cheating: 1/6):
- Party 1: \( x_1 \)  bcasts: \( \alpha_1, \beta_1, \ldots, \)result_1
- Party 2: \( x_2 \)  bcasts: \( \alpha_2, \beta_2, \ldots, \)result_2
- Party 3: \( x_3 \)  bcasts: \( \alpha_3, \beta_3, \ldots, \)result_3
- Party 4: \( x_4 \)  bcasts: \( \alpha_4, \beta_4, \ldots, \)result_4
- Party 5: \( x_5 \)  bcasts: \( \alpha_5, \beta_5, \ldots, \)result_5
- Party 6: \( x_6 \)  bcasts: \( \alpha_6, \beta_6, \ldots, \)result_6
Partitioning the parties - Sub-MPC protocols

Plaintext Protocol:
Plaintext: $x_1 + \cdots + x_6$ \hspace{1cm} plain bcasts: $\alpha, \beta, \ldots, \text{result}$

Original 6-players Protocol (chances of cheating: 1/6):
- Party 1: $x_1$ \hspace{1cm} bcasts: $\alpha_1, \beta_1, \ldots, \text{result}_1$
- Party 2: $x_2$ \hspace{1cm} bcasts: $\alpha_2, \beta_2, \ldots, \text{result}_2$
- Party 3: $x_3$ \hspace{1cm} bcasts: $\alpha_3, \beta_3, \ldots, \text{result}_3$
- Party 4: $x_4$ \hspace{1cm} bcasts: $\alpha_4, \beta_4, \ldots, \text{result}_4$
- Party 5: $x_5$ \hspace{1cm} bcasts: $\alpha_5, \beta_5, \ldots, \text{result}_5$
- Party 6: $x_6$ \hspace{1cm} bcasts: $\alpha_6, \beta_6, \ldots, \text{result}_6$

Red Sub Protocol (chances of cheating: 1/2):
- Group 1: $x_1 + x_2 + x_3$ \hspace{1cm} bcasts: $\alpha_1, \beta_1, \ldots, \text{result}_1$
- Group 2: $x_4 + x_5 + x_6$ \hspace{1cm} bcasts: $\alpha_2, \beta_2, \ldots, \text{result}_2$
Partitioning the parties - Sub-MPC protocols

<table>
<thead>
<tr>
<th>Party</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
</tr>
</thead>
</table>

Plaintext Protocol:

Plaintext:  $x_1 + \cdots + x_6$

plain bcasts: $\alpha, \beta, \ldots, \text{result}$

Original 6-players Protocol (chances of cheating: $1/6$):

- Party 1: $x_1$
- Party 2: $x_2$
- Party 3: $x_3$
- Party 4: $x_4$
- Party 5: $x_5$
- Party 6: $x_6$

Red Sub Protocol (chances of cheating: $1/2$):

- Group 1: $x_1 + x_2 + x_3$
- Group 2: $x_4 + x_5 + x_6$

Blue Sub Protocol (chances of cheating: $1/3$):

- Group 1: $x_1 + x_4$
- Group 2: $x_2 + x_5$
- Group 3: $x_3 + x_6$
Partitioning the parties - Sub-MPC protocols

Plaintext Protocol:

Plaintext: \(x_1 + \cdots + x_6\)

Original 6-players Protocol (chances of cheating: 1/6):

<table>
<thead>
<tr>
<th>Party 1</th>
<th>Party 2</th>
<th>Party 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(x_3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Party 4</th>
<th>Party 5</th>
<th>Party 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_4)</td>
<td>(x_5)</td>
<td>(x_6)</td>
</tr>
</tbody>
</table>

bcasts: \(\alpha, \beta, \ldots, \text{result}\)

Red Sub Protocol (chances of cheating: 1/2):

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1 + x_2 + x_3)</td>
<td>(x_4 + x_5 + x_6)</td>
</tr>
</tbody>
</table>

bcasts: \(\alpha_1, \beta_1, \ldots, \text{result}_1\)

bcasts: \(\alpha_2, \beta_2, \ldots, \text{result}_2\)

Blue Sub Protocol (chances of cheating: 1/3):

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1 + x_4)</td>
<td>(x_2 + x_5)</td>
<td>(x_3 + x_6)</td>
</tr>
</tbody>
</table>

bcasts: \(\alpha_1, \beta_1, \ldots, \text{result}_1\)

bcasts: \(\alpha_2, \beta_2, \ldots, \text{result}_2\)

bcasts: \(\alpha_3, \beta_3, \ldots, \text{result}_3\)

independent!!
Partitioning the parties - Sub-MPC protocols

Plaintext Protocol:
Plaintext: $x_1 + \cdots + x_6$ plain bcasts: $\alpha, \beta, \ldots$, result

Original 6-players Protocol (chances of cheating: $1/6$):
- Party 1: $x_1$
- Party 2: $x_2$
- Party 3: $x_3$
- Party 4: $x_4$
- Party 5: $x_5$
- Party 6: $x_6$

Red Sub Protocol (chances of cheating: $1/2$):
- Group 1: $x_1 + x_2 + x_3$
- Group 2: $x_4 + x_5 + x_6$

Blue Sub Protocol (chances of cheating: $1/3$):
- Group 1: $x_1 + x_4$
- Group 2: $x_2 + x_5$
- Group 3: $x_3 + x_6$
Faster and Smaller proofs: pushing the tradeoff

Single MPC-in-the-head instance: $\log_2(n)$ bits of security
- Faster MPC-in-the-head that preserve soundness and small proof size
- Within the previous running time, we can take $n$ larger

Parallel composition to achieve $\lambda$ bits of security
- Less parallel repetitions to achieve $1/2^\lambda$ security $\implies$ smaller and faster.

Fiat-Shamir Transform
- HVZK proof with small communications $\implies$ Small signature.
Part II - Hypercube SD-in-the-Head
The inhomogeneous SD problem

Given \( H = (\text{Id}_{m-k} \| H') \) a random \( m \times m - k \) matrix over \( \mathbb{F}_q \), and a random syndrom \( y \in \mathbb{F}_q^{m-k} \), find a solution \( x \in \mathbb{F}_q^m \) of:

\[
Hx = y \text{ where hamming weight}(x) \leq w
\]
Equivalent formulation of the ISD problem

Given $H'$ and $y$, find one vector $x_A \in \mathbb{F}_q^k$ and one polynomials $Q \in \mathbb{F}_q[X]$ monic of degree $w$ and $P(X)$ of degree $\leq w - 1$ such that

$$Q \times \text{interpolation}_{[1,m]}(x_A || (y - H'x_A)) - P \times (X - 1)...(X - m) = 0$$

Randomized verification function (w. false positive proba $p$)

Evaluate the above polynomial in MPC over just one random verifier-supplied point (in an extension field if needed). If the result is zero, the proof is accepted.

Soundness of 1 iteration of SDitH: $(1 - p) \left(1 - \frac{1}{N}\right)$
**SD Verification in MPC**

**Equivalent formulation of the ISD problem**

Given $H'$ and $y$, find one vector $x_A \in \mathbb{F}_q^k$ and one polynomials $Q \in \mathbb{F}_q[X]$ monic of degree $w$ and $P(X)$ of degree $\leq w - 1$ such that

$$Q \times \text{interpolation}_{[1,m]}(x_A || (y - H'x_A)) - P \times (X - 1)...(X - m) = 0$$

**Randomized verification function (w. false positive proba $p$)**

Evaluate the above polynomial in MPC over just one random verifier-supplied point (in an extension field if needed). If the result is zero, the proof is accepted.

Soundness of 1 iteration of SDitH: $(1 - p) \left(1 - \frac{1}{N}\right)$
The SD and MPC parameters for our protocol, originally from [FJR22].

<table>
<thead>
<tr>
<th>Scheme</th>
<th>SD Parameters</th>
<th>MPC Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$q$</td>
<td>$m$</td>
</tr>
<tr>
<td>Variant 1</td>
<td>2</td>
<td>1280</td>
</tr>
<tr>
<td>Variant 2</td>
<td>2</td>
<td>1536</td>
</tr>
<tr>
<td>Variant 3</td>
<td>$2^8$</td>
<td>256</td>
</tr>
</tbody>
</table>
Our parameters with key and signature sizes in bytes for $\lambda = 128$.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Aim</th>
<th>Parameters</th>
<th>Sizes (in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$N$</td>
<td>$D$</td>
</tr>
<tr>
<td>Variant 3</td>
<td>Fast</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Short</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Shorter</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Shortest</td>
<td>2</td>
<td>16</td>
</tr>
</tbody>
</table>
Benchmarks and performance of Hypercube-SDitH

Table 7: Reference implementation benchmarks of SDitH [FJR22] vs our scheme for $\lambda = 128$. Both ran on a single CPU core of a 3.1 GHz Intel Core i9-9900K.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Aim</th>
<th>Signature Size</th>
<th>Parameters $N \quad D \quad \tau$</th>
<th>Sign Time (in ms)</th>
<th>Verify Time (in ms) Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDitH [FJR22]</td>
<td>Fast</td>
<td>12 115</td>
<td>32 - 27</td>
<td>0.87 5.03 5.96</td>
<td>4.74</td>
</tr>
<tr>
<td></td>
<td>Short</td>
<td>8 481</td>
<td>256 - 17</td>
<td>4.33 18.95 23.56</td>
<td>20.80</td>
</tr>
<tr>
<td>(Variant 3)</td>
<td>Shorter</td>
<td>6 784</td>
<td>$2^{12}$ - 12</td>
<td>59.24 251.14 313.70</td>
<td>244.30</td>
</tr>
<tr>
<td></td>
<td>Shortest</td>
<td>5 689</td>
<td>$2^{16}$ - 9</td>
<td>- - -</td>
<td>-</td>
</tr>
<tr>
<td>Ours (Variant 3)</td>
<td>Fast</td>
<td>12 115</td>
<td>2 5 27</td>
<td>0.47 0.83 1.30</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>Short</td>
<td>8 481</td>
<td>2 8 17</td>
<td>2.26 0.61 2.87</td>
<td>2.59</td>
</tr>
<tr>
<td></td>
<td>Shorter</td>
<td>6 784</td>
<td>2 12 12</td>
<td>25.93 0.50 26.43</td>
<td>25.79</td>
</tr>
<tr>
<td></td>
<td>Shortest</td>
<td>5 689</td>
<td>2 16 9</td>
<td>320.24 0.42 320.66</td>
<td>312.67</td>
</tr>
</tbody>
</table>
Conclusion and perspectives

A new post quantum signature candidate for NIST (WIP)

- Hypercube-SDitH in the QROM model (vs. ROM)
- Parameters suitable for $\lambda = 128, 192$ and $256$
- SD over prime fields
- Hypercube-SDitH with other tradeoffs (e.g. Threshold-SDitH)

Other goodies

- Microsecond latency: Offline/Online phase model?
- Applications to other hard problems?

Open problem / Limitation

- State generation is still in $O(n)$: we cannot take $n$ exponential
Thank you!