Signatures from identification, MPCitH, and more

Andreas Hülsing
Eindhoven University of Technology & SandboxAQ

PQ Signatures

- Signatures vs KEM: Should be easier … it isn't...
- Approaches:
  1. Hash & Sign
     - Full Domain Hash (FDH) with Trapdoor OWP: RSA-PSS, MAYO, UOV,...
     - FDH with Preimage-sampleable TDF: Falcon
     - Hash-based signatures
  2. Signatures from identification:
     - Fiat-Shamir (FS): (EC)DSA, Schnorr, …
     - FS with aborts: Dilithium
     - FS + MPC in the Head (MPCitH): Picnic, Biscuit, MIRA, MiRitH, MQOM, PERK, RYDE, SDitH, AIMer, ...

Syndrome Decoding in the Head (FJR22)

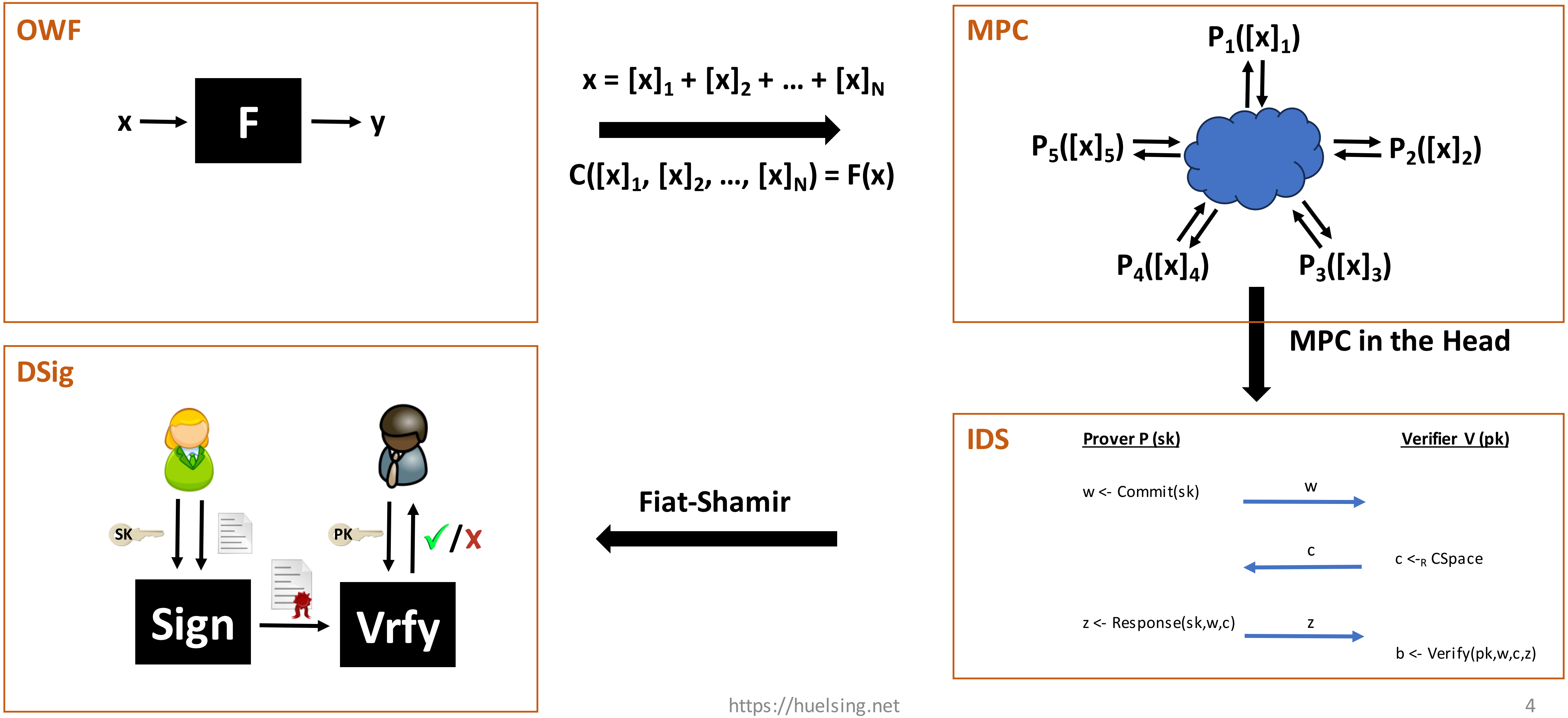
- Code-based signature scheme using MPCitH
- Beats all previous code-based signatures
- Uses unstructured SD problem!

Scheme Name	Year	|sgn|	|pk|	t_{sgn}	t_{verif}	Assumption
Wave	2019	2.07 K	3.2 M	300	-	SD over $\mathbb{F}_3$ (large weight) $(U, U+V)$-codes indisting.
Durandal - I	2018	3.97 K	14.9 K	4	5	Rank SD over $\mathbb{F}_{2^m}$
Durandal - II	2018	4.90 K	18.2 K	5	6	Rank SD over $\mathbb{F}_{2^m}$
LESS-FM - I	2020	15.2 K	9.77 K	-	-	Linear Code Equivalence
LESS-FM - II	2020	5.25 K	206 K	-	-	Perm. Code Equivalence
LESS-FM - III	2020	10.39 K	11.57 K	-	-	Perm. Code Equivalence
[GPS22]-256	2021	24.0 K	0.11 K	-	-	SD over $\mathbb{F}_{256}$
[GPS22]-1024	2021	19.8 K	0.12 K	-	-	SD over $\mathbb{F}_{1024}$
[FJR21] (fast)	2021	22.6 K	0.09 K	13	12	SD over $\mathbb{F}_2$
[FJR21] (short)	2021	16.0 K	0.09 K	62	57	SD over $\mathbb{F}_2$
[BGKM22] - Sig1	2022	23.7 K	0.1 K	-	-	SD over $\mathbb{F}_2$
[BGKM22] - Sig2	2022	20.6 K	0.2 K	-	-	(QC)SD over $\mathbb{F}_2$
Our scheme - Var1f	2022	15.6 K	0.09 K	-	-	SD over $\mathbb{F}_2$
Our scheme - Var1s	2022	10.9 K	0.09 K	-	-	SD over $\mathbb{F}_2$
Our scheme - Var2f	2022	17.0 K	0.09 K	13	13	SD over $\mathbb{F}_2$
Our scheme - Var2s	2022	11.8 K	0.09 K	64	61	SD over $\mathbb{F}_2$
Our scheme - Var3f	2022	11.5 K	0.14 K	6	6	SD over $\mathbb{F}_{256}$
Our scheme - Var3s	2022	8.26 K	0.14 K	30	27	SD over $\mathbb{F}_{256}$

Table 6. Comparison of our scheme with signatures from the literature (128-bit security). The sizes are in bytes and the timings are in milliseconds. Reported timings are from the original publications: Wave has been benchmarked on a 3.5 Ghz Intel Xeon E3-1240 v5, Durandal on a 2.8 Ghz Intel Core i5-7440HQ, while [FJR21] and our scheme on a 3.8 GHz Intel Core i7.

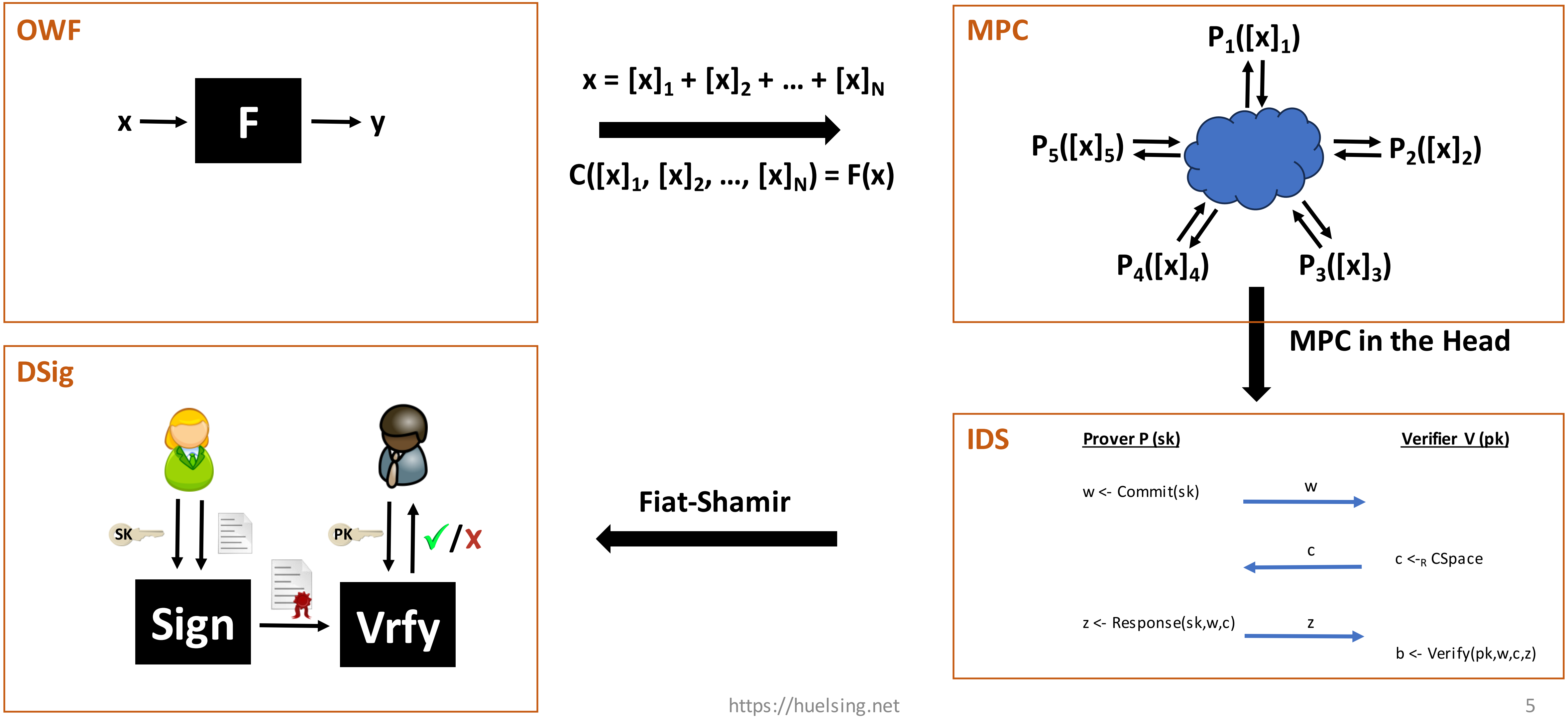
Source:
Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. *"Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs".* Crypto'22

Outline



OWF
x
F
y
$x = [x]_1 + [x]_2 + \ldots + [x]_N$
$C([x]_1, [x]_2, \ldots, [x]_N) = F(x)$
MPC
$P_1([x]_1)$
$P_2([x]_2)$
$P_3([x]_3)$
$P_4([x]_4)$
$P_5([x]_5)$
MPC in the Head
IDS
Prover P (sk)
Verifier V (pk)
w <- Commit(sk)
w
c
$c \leftarrow_R$ CSpace
z <- Response(sk,w,c)
z
b <- Verify(pk,w,c,z)
Fiat-Shamir
DSig
SK
PK
✓/X
Sign
Vrfy

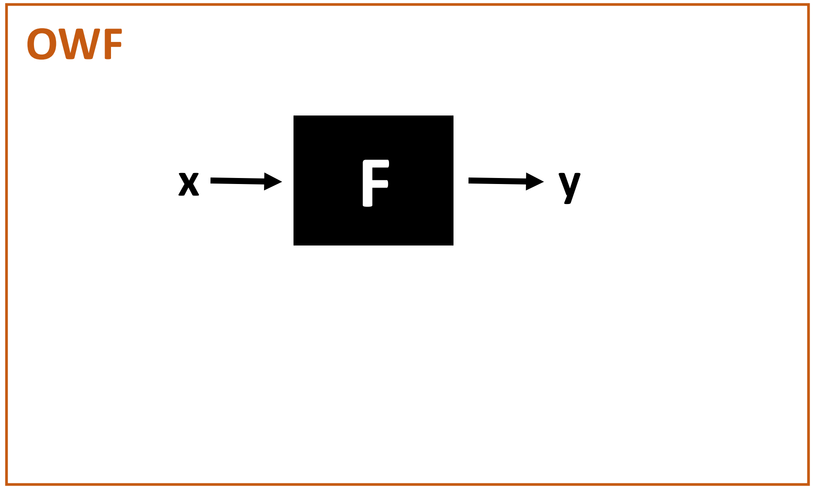
Outline

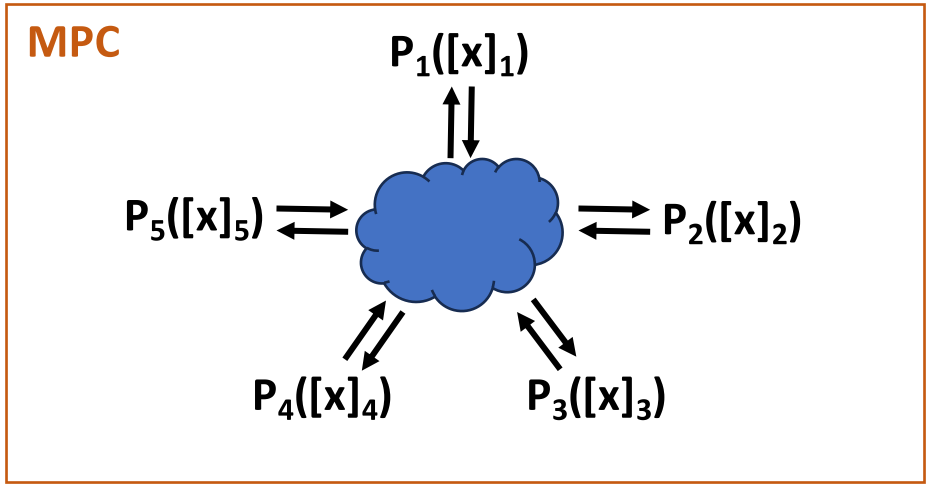

OWF
x
F
y
x = [x]1 + [x]2 + … + [x]N
C([x]1, [x]2, …, [x]N) = F(x)
MPC
P1([x]1)
P2([x]2)
P3([x]3)
P4([x]4)
P5([x]5)
MPC in the Head
IDS
Prover P (sk)
Verifier V (pk)
w <- Commit(sk)
w
c
c <-R CSpace
z <- Response(sk,w,c)
z
b <- Verify(pk,w,c,z)
Fiat-Shamir
DSig
SK
PK
Sign
Vrfy
✓/X

OWF

- AES (FEAST)
- LowMC (Picnic)
- AIM (AIMer)
- Polynomial arithmetic & evaluation (SDitH)
- MQ equation system (Biscuit)

Low multiplicative depth is an advantage!



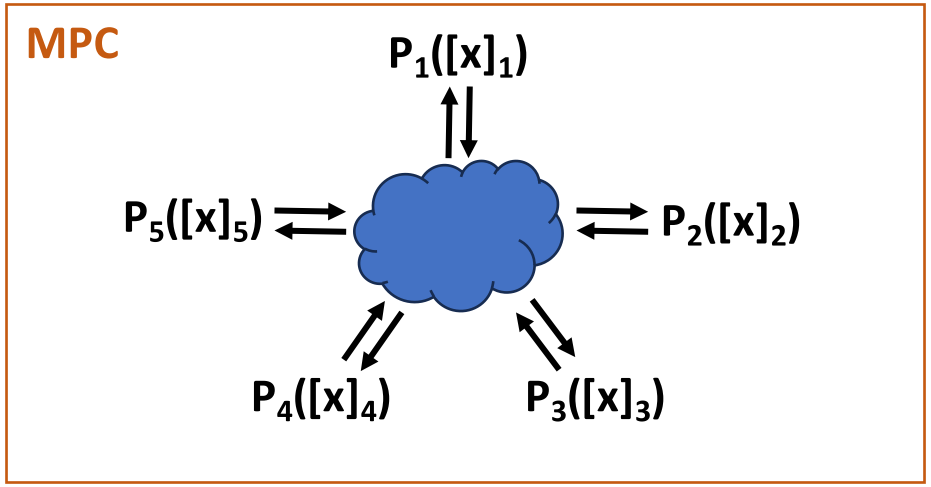


(secure) Multi-Party Computation

MPC

Allows N parties $P_1, \ldots, P_N$ with inputs $x_1, \ldots, x_N$ to jointly compute a function $F(x_1, \ldots, x_N) = y$ such that

- all parties learn the outcome y
- but nothing beyond that



Example: Price negotiations

Buyer & Seller compute if they can agree on price X

- Logical AND of "willingness"
- If you do not agree, you do not learn the other party's decision!
- Prevents pushing up / down to limit of other party

MPC

Allows N parties $P_1, \dots, P_N$ with inputs $x_1, \dots, x_N$ to jointly compute a function $F(x_1, \dots, x_N) = y$ such that

- all parties learn the outcome y
- but nothing beyond that

We additionally need:

- **Correctness:** If all parties are honest, the result is correct
- **N-1 private:** If N-1 parties collaborate, they can still not learn anything about the input of the last party beyond what can be derived from $F(x_1, \dots, x_N) = y$
- **Broadcast communication:** All messages are broadcasted

(additive) Secret Sharing Scheme (SSS)

Split $x = [x]_1 + [x]_2 + \ldots + [x]_N$ with secret shares $[x]_i$ in $\mathbb{F}_q$

- Given all but one share x is information theoretically hidden!

MPC for additive SSS

Split $x = [x]_1 + [x]_2 + \ldots + [x]_N$ with secret shares $[x]_i$ in $\mathbb{F}_q$

Party i holds share $[x]_i$ of value x.

Operations:

- Adding shared values ($[x] + [y]$): Parties locally add shares
  $\Sigma([x]_i+[y]_i) = \Sigma[x]_i + \Sigma[y]_i = x+y$
- Adding constant ($[x] + c$): P1 computes $[x]_1 + c$, all others do nothing
  $[x]_1 + c + [x]_2 + \ldots + [x]_N = \Sigma[x]_i + c = x + c$
- Multiplication by constant ($[x] \cdot c$): All parties locally compute $[x]_i \cdot c$
  $[x]_1 \cdot c + [x]_2 \cdot c + \ldots + [x]_N \cdot c = ([x]_1 + [x]_2 + \ldots + [x]_N) \cdot c = x \cdot c$

MPC for additive SSS

Split $x = [x]_1 + [x]_2 + \ldots + [x]_N$ with secret shares $[x]_i$ in $\mathbb{F}_q$

Party i holds share $[x]_i$ of value x.

Operations:

- Adding shared values ($[x] + [y]$): Parties locally add shares
  $\Sigma([x]_i+[y]_i) = \Sigma[x]_i+ \Sigma[y]_i = x+y$
- Adding constant ($[x] + c$): P1 computes $[x]_1 + c$, all others do nothing
  $[x]_1 + c + [x]_2 + \ldots + [x]_N = \Sigma[x]_i + c = x + c$
- Multiplication by constant ($[x] \cdot c$): All parties locally compute $[x]_i \cdot c$
  $[x]_1 \cdot c + [x]_2 \cdot c + \ldots + [x]_N \cdot c = ([x]_1 + [x]_2 + \ldots + [x]_N) \cdot c = x \cdot c$

Multiplication of shared values?

Share multiplication

- Conventional:
  - (Katz, Kolesnikov, Wang. "*Improved non-interactive zero knowledge with applications to post-quantum signatures*". CCS 2018)
  - All parties know one share of both inputs
  - After protocol, all parties know a share of the output

- Modern:
  - (Lindell, Nof. "*A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority*". CCS 2017)
  - (Baum, Nof. "*Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography*". PKC 2020)
  - All parties know a share of both inputs and the output
  - Protocol proves that output is a sharing of product of input

Verifying multiplication

Parties need random triple [a], [b], [c], with ab = c,
to verify [x], [y], [z], with xy = z

- Take random element e in F_q
- Parties locally set [α] = e[x] + [a] and [β] = [y] + [b]
- Parties broadcast [α] and [β] shares to open α and β
- Parties locally set [v] = e[z] − [c] + α · [b] + β · [a] − α · β (note that last summand is only subtracted by P_1)
- Parties broadcast [v] shares to open v and accept if v = 0.

Verifying multiplication – Correctness

- $v = e \cdot z - c + \alpha \cdot b + \beta \cdot a - \alpha \cdot \beta$

  $= e \cdot xy - ab + (e \cdot x + a)b + (y + b)a - (e \cdot x + a)(y + b)$

  $= exy - ab + exb + ab + ya + ba - exy - exb - ay - ab = 0$

Verifying multiplication – Soundness

- $v = e \cdot z - c + \alpha \cdot b + \beta \cdot a - \alpha \cdot \beta$

  $= e \cdot xy - ab + (e \cdot x + a)b + (y + b)a - (e \cdot x + a)(y + b)$

  $= exy - ab + exb + ab + ya + ba - exy - exb - ay - ab = 0$

Verifying multiplication – Soundness

- Let $z = xy + d_z$ and $c = ab + d_c$
- $v = e \cdot z - c + \alpha \cdot b + \beta \cdot a - \alpha \cdot \beta$

  $= e \cdot xy + ed_z - ab - d_c + (e \cdot x + a)b + (y + b)a - (e \cdot x + a)(y + b)$

  $= 0 + ed_z - d_c$

Verifying multiplication – Soundness

Claim: If $d_z \neq 0$ or $d_c \neq 0$ then $v = 0$ with probability at most $1 / |F_q|$

Proof: Recall $v = ed_z - d_c$

- Case $d_z = 0$ & $d_c \neq 0$:

$$v = ed_z - d_c = - d_c \neq 0$$

Verifying multiplication – Soundness

Claim: If $d_z \neq 0$ or $d_c \neq 0$ then $v = 0$ with probability at most $1 / |F_q|$

Proof: Recall $v = ed_z - d_c$

- Case $d_z = 0$ & $d_c \neq 0$:

$$v = ed_z - d_c = - d_c \neq 0$$

- Case $d_z \neq 0$ & $d_c \neq 0$:

$$v = 0 \quad \Leftrightarrow \quad d_c = ed_z \quad \Leftrightarrow \quad d_c d_z^{-1} = e \quad (\text{prob } 1 / |F_q|)$$

Verifying multiplication – Soundness

Claim: If $d_z \neq 0$ or $d_c \neq 0$ then $v = 0$ with probability at most $1 / |F_q|$

Proof: Recall $v = ed_z - d_c$

- Case $d_z = 0$ & $d_c \neq 0$:

$$v = ed_z - d_c = - d_c \neq 0$$

- Case $d_z \neq 0$ & $d_c \neq 0$:

$$v = 0 \quad \Leftrightarrow \quad d_c = ed_z \quad \Leftrightarrow \quad d_c d_z^{-1} = e \quad (\text{prob } 1 / |F_q|)$$

- Case $d_z \neq 0$ & $d_c = 0$:

$$v = ed_z - d_c = edz \quad \Rightarrow \quad v = 0 \text{ iff } e = 0 \quad (\text{prob } 1 / |F_q|)$$

Function to circuit - Examples

Evaluating shared polynomial $[P] = \Sigma [p_i] x^i$ at public point r:

- Locally: $[P](r) = \Sigma [p_i] r^i = [y]$
  - No interaction
  - Single secret shared value as outcome

Evaluating product of shared polynomials [P], [S] at public point r:

- Requires knowledge of result [z]
- Locally: $[P](r) = \Sigma [p_i] r^i = [y]$, $[S](r) = \Sigma [s_i] r^i = [x]$
- Run verify for $[x]\cdot[y] = [z]$
  - Single broadcast interaction + final opening

Function to circuit: SDitH (FJR'22)

- Turn Syndrome Decoding function into MPC

Definition 4 (Coset Weights Syndrome Decoding problem). *Sample a uniformly random parity check matrix* $\mathbf{H} \in \mathbb{F}_{SD}^{(m-k)\times m}$, *and binary vector* $\mathbf{x} \in \mathbb{F}_{SD}^{m}$ *with* $wt(\mathbf{x}) = \omega$. *Let syndrome* $\mathbf{y} = \mathbf{Hx}$. *Then given only* $\mathbf{H}, \mathbf{y}$, *it is difficult to find* $\mathbf{x}' \in \mathbb{F}_{SD}^{m}$ *such that* $\mathbf{Hx}' = \mathbf{y}$ *with* $wt(\mathbf{x}') \leq \omega$.

Function to circuit: SDitH (FJR'22)

- Turn Syndrome Decoding function into MPC

Definition 4 (Coset Weights Syndrome Decoding problem). *Sample a uniformly random parity check matrix* $\mathbf{H} \in \mathbb{F}_{SD}^{(m-k)\times m}$, *and binary vector* $\mathbf{x} \in \mathbb{F}_{SD}^{m}$ *with* $wt(\mathbf{x}) = \omega$. *Let syndrome* $\mathbf{y} = \mathbf{H}\mathbf{x}$. *Then given only* $\mathbf{H}, \mathbf{y}$, *it is difficult to find* $\mathbf{x}' \in \mathbb{F}_{SD}^{m}$ *such that* $\mathbf{H}\mathbf{x}' = \mathbf{y}$ *with* $wt(\mathbf{x}') \leq \omega$.

- Advantage: Linear function.

Function to circuit: SDitH (FJR'22)

- Turn Syndrome Decoding function into MPC

Definition 4 (Coset Weights Syndrome Decoding problem). *Sample a uniformly random parity check matrix* $\mathbf{H} \in \mathbb{F}_{SD}^{(m-k)\times m}$*, and binary vector* $\mathbf{x} \in \mathbb{F}_{SD}^{m}$ *with* $wt(\mathbf{x}) = \omega$*. Let syndrome* $\mathbf{y} = \mathbf{H}\mathbf{x}$*. Then given only* $\mathbf{H}, \mathbf{y}$*, it is difficult to find* $\mathbf{x}' \in \mathbb{F}_{SD}^{m}$ *such that* $\mathbf{H}\mathbf{x}' = \mathbf{y}$ *with* $wt(\mathbf{x}') \leq \omega$*.*

- Advantage: Linear function.
- Disadvantage: Weight check.

SDitH – Implicit Equation Check

- Use H in standard form: $H = (H' \mid I_{m-k})$
- Can write $x = (x_A \mid x_B)$ with $y = H'x_A + x_B$
- Define $sk = x_A$
- Compute x via $x_B = y - H'x_A$

  => guarantees x fulfills $y = Hx$

SDitH – Weight check

- Compute x from x_A, H, and y
- Derive a polynomial S from x
- Generate polys Q, P, and public F such that

$$SQ - PF = 0 \quad \text{if } wt(x) \leq \omega.$$

- Select t random points r_i and verify that

$$S(r_i)Q(r_i) = PF(r_i) \text{ for } 0 < i \leq t.$$

SDitH – MPC circuit

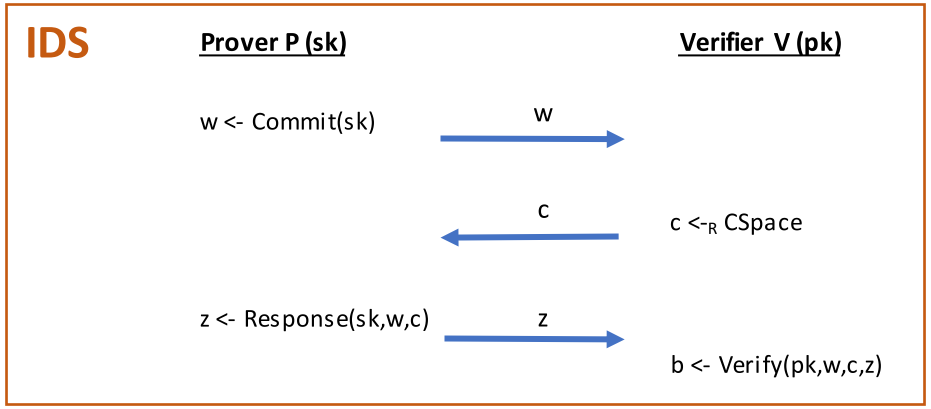
- Compute [x] from [x_A], H, and y (only linear ops)
- Derive share of polynomial [S] from [x] (only linear ops)
- Generate secret shared polys [Q], [P], and public F such that

$$[S][Q] - [P]F = 0 \text{ if } wt(x) \leq \omega.$$

- Get t random points r_i, t random masks e_i, and run verification for

$$[S](r_i)[Q](r_i) = [P]F(r_i) \text{ using } e_i$$

for $0 < i \leq t$.



Identification Schemes

Identification Schemes (IDS) / Zero-knowledge proofs (ZKP)

- Invented by Shafi Goldwasser, Silvio Micali and Charles Rackoff in 1985
- Interactive proof systems
- Prove knowledge of a secret without revealing any information about the secret
- [For people that like classifications: The IDS we discuss are actually Honest-Verifier Zero-Knowledge Arguments of Knowledge]

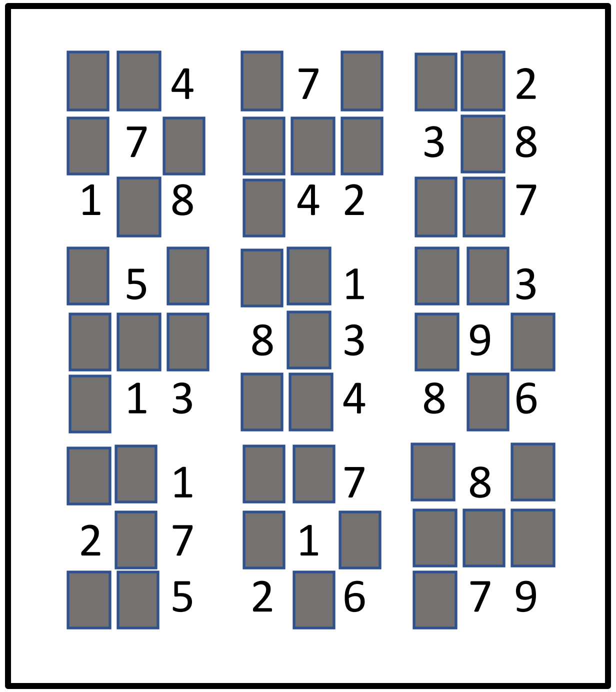
Identification schemes (3-round, public coin)

Prover P (sk)		**Verifier V (pk)**
w <- Commit(sk)	w $\longrightarrow$	
	c $\longleftarrow$	c <-$_R$ CSpace
z <- Response(sk,w,c)	z $\longrightarrow$	
		b <- Verify(pk,w,c,z)

Also called a "Sigma Protocol"

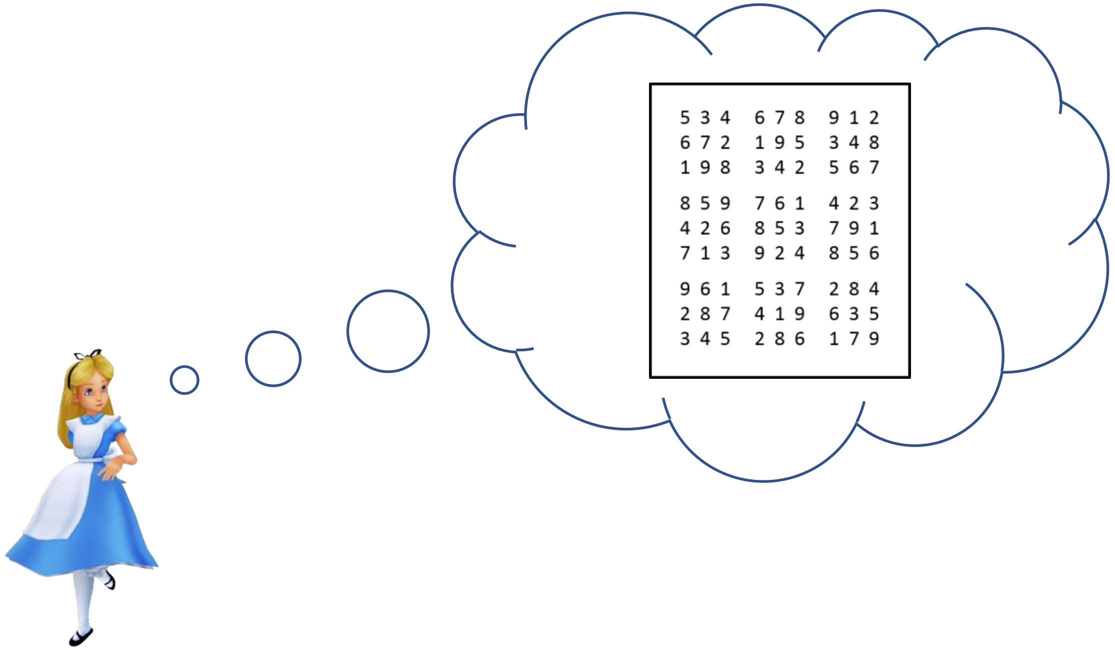
The case of Sudoku

- A: I have a nice Sudoku for you
- B: You are sure this is solvable?
- A: Sure!
- B: Prove it!
- A: Ok…



The case of Sudoku

- So how can Alice prove that a solution exists without making the Sudoku easier (a.k.a. leaking information)?



The case of Sudoku

- Apply random permutation to solution:

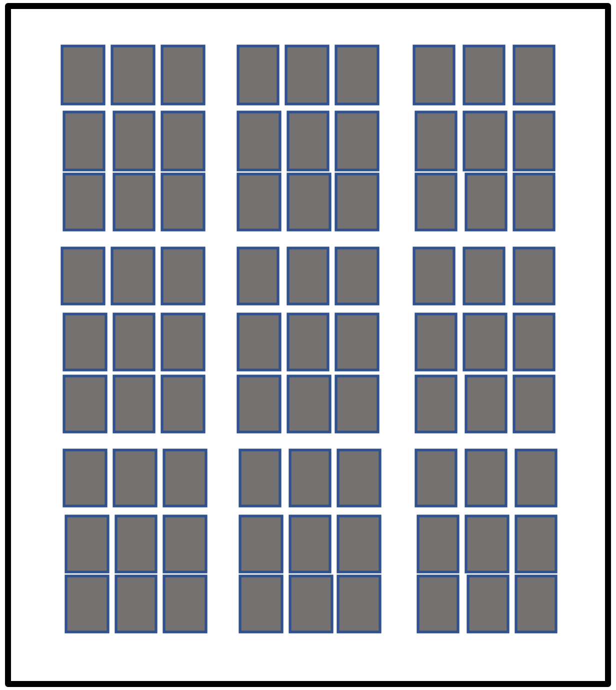
1	2	3	4	5	6	7	8	9
3	2	7	1	6	9	4	5	8

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

6	7	1	9	4	5	8	3	2
9	4	2	3	8	6	7	1	5
3	8	5	7	1	2	6	9	4
5	6	8	4	9	3	1	2	7
1	2	9	5	6	7	4	8	3
4	3	7	8	2	1	5	6	9
8	9	3	6	7	4	2	5	1
2	5	4	1	3	8	9	7	6
7	1	6	2	5	9	3	4	8

The case of Sudoku

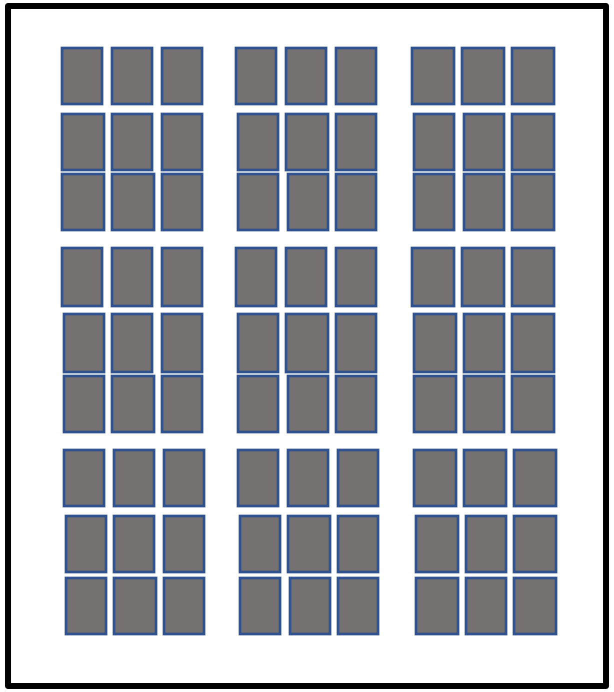
- Prepare scratch card:



The case of Sudoku

Show scratch card to Bob and allow him to ask Alice to do **one** out of the following:

- Scratch off a row
- Scratch off a column
- Scratch off a square
- Scratch off original Sudoku



The case of Sudoku

What does Bob gain? (Soundness)

- If scratching reveals inconsistency:
  Alice cheated!
- If scratching reveals consistent values:
  Alice might have cheated...

  But Bob gains some confidence in Alice
  knowing a solution.

The case of Sudoku

- Bob choose from 28 possible “challenges“
- If Alice is cheating she gets caught with prob. $\geq \frac{1}{28}$
- Cheating Alice has chance of $\leq \frac{27}{28}$ to succeed
- Repeating protocol n times means Alice’s cheating probability goes down to

$$\left(\frac{27}{28}\right)^{n} \approx \left(\frac{1}{2}\right)^{0.05n}$$

- When $n = 2500$, Alice caught with 0.99 probability.

The case of Sudoku

(Honest-Verifier) Zero-knowledge:

- We want to show that (honest) Bob does not learn anything about the secret (i.e., the Sudoku solution)
- We will prove: Everything he learns, he could have generated himself.
- Can be proved showing that Bob (without knowing the secret) could have generated valid protocol transcripts that are indistinguishable from those obtained by communicating with Alice.

The case of Sudoku

Proving zero-knowledge:

- Trick: When Bob generates transcripts, he can first select the challenge, then produce the scratch card!

- For challenge row, column, or square: Just put random permutation of 1...9.
- For challenge original Sudoku: Just put random permutation of the used numbers.

$\Longrightarrow$ Follows exactly same distribution as what Alice would have put there!

The case of Sudoku - Implications

Yato 2003:
„Solvability of n x n Sudoku is NP-complete"

- We can use this proof for any other problem in NP
- Just transform problem instance into Sudoku instance and run ZKP for that instance.

Identification schemes (3-round, public coin)

Prover P (sk)		**Verifier V (pk)**
w <- Commit(sk)	$\xrightarrow{w}$	
	$\xleftarrow{c}$	c <-$_R$ CSpace
z <- Response(sk,w,c)	$\xrightarrow{z}$	
		b <- Verify(pk,w,c,z)

Also called a "Sigma Protocol"

Security Properties

- **Soundness:** A prover that does not know the secret will get caught with high probability (1 – e) where e is called soundness error
- **Special soundness:** There exists an efficient extractor E that given two transcripts with same w but different c, extracts sk.
- **Honest verifier zero-knowledge (HVZK):** There exists an efficient simulator S that, given only the public key, outputs transcripts which are indistinguishable from transcripts of honest protocol runs

Identification schemes (5-round, public coin)

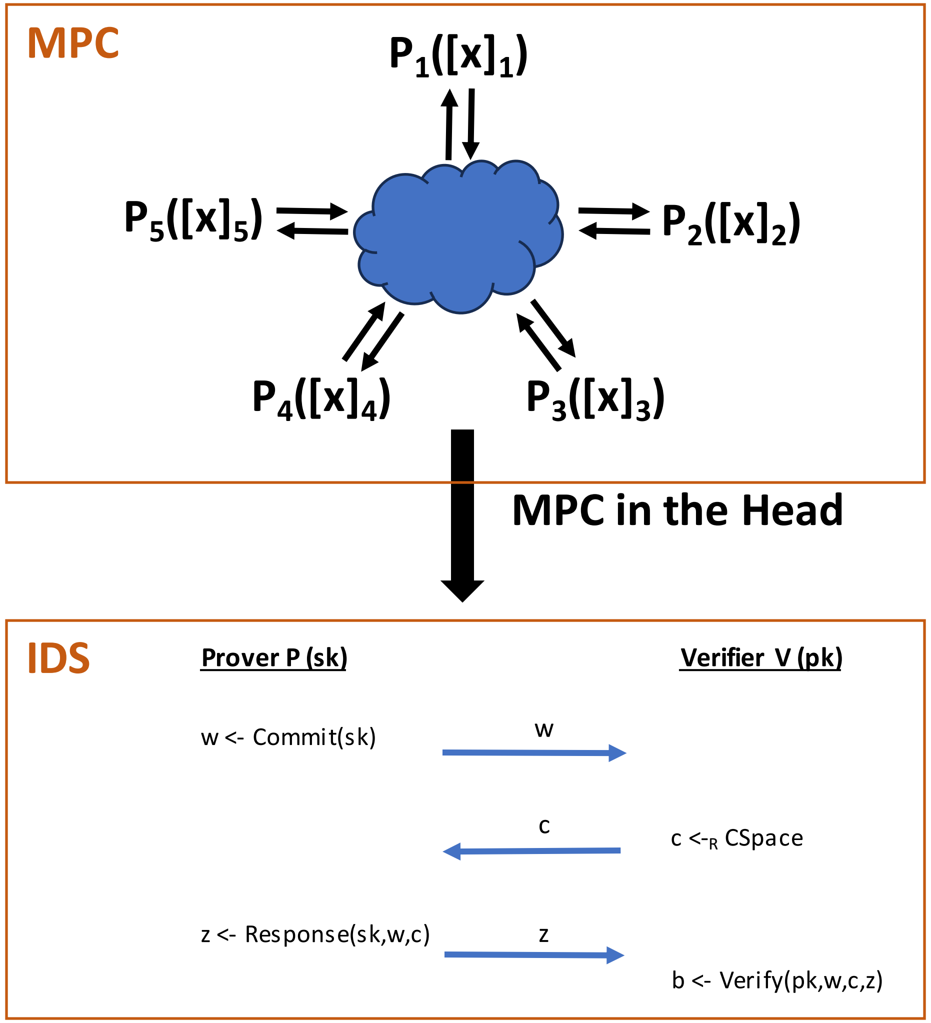
Prover P (sk)		**Verifier V (pk)**
w <- Commit(sk)	w	
	c_1	c_1 <-$_R$ $CSpace_1$
z_1 <- Response(sk,w,c_1)	z_1	
	c_2	c_2 <-$_R$ $CSpace_2$
z_2 <- Response(sk,w,c_1,z_1,c_2)	z	
		b <- Verify(pk,w,c_1,z_1,c_2,z_2)

More notes on IDS

- We can have 2n+1 round IDS for $n \geq 1$
- We usually require that w has high entropy (hard to predict)
- Commitment-recoverable IDS:
  - There exist function Recv(c, z) -> w
- We later need negligible soundness error
  - Achieved via parallel composition

MPC in the Head

Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai.
“Zero-knowledge from secure multiparty computation”. STOC'07



46

MPCitH for PQ-identification

(Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. *"Zero-knowledge from secure multiparty computation"*. STOC'07)

Given OWF F: X -> Y

Create identification scheme IDS that proves knowledge of x such that

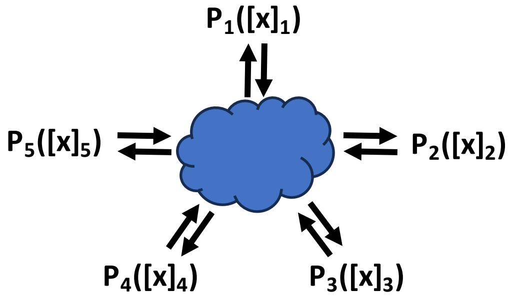
$$F(x) = y$$

for given y in (honest-verifier) zero-knowledge.

sk = x, pk = y[, F]

High-level idea

Prover P (sk) | **Verifier V (pk)**



w <- Commit([x]),  $\xrightarrow{\quad w \quad}$

$\xleftarrow{\quad c \quad}$ c $<-_R$ {1, … , 5}

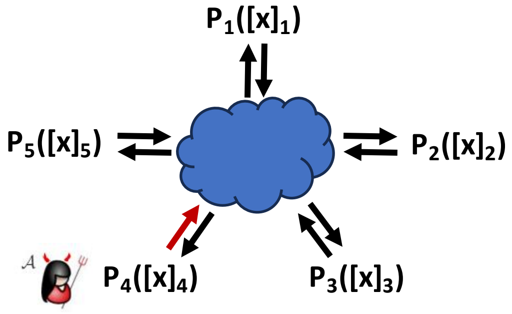
z <- P_i forall i ≠ c $\xrightarrow{\quad z \quad}$

Check consistency of all opened parties & verify result

Security - Soundness

Prover P (sk)

Verifier V (pk)



w <- Commit([x]),

w →

← c

$c \leftarrow_R \{1, \dots, 5\}$

z <- P_i forall $i \neq c$

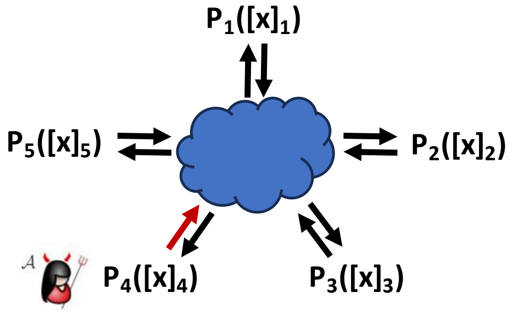
z →

Check consistency of all opened parties & verify result

Security - Soundness

Prover P (sk)

Verifier V (pk)



w <- Commit([x]), 

$\xrightarrow{\quad w \quad}$

$\xleftarrow{\quad c = 4 \quad}$

$c \leftarrow_R \{1, \ldots, 5\}$

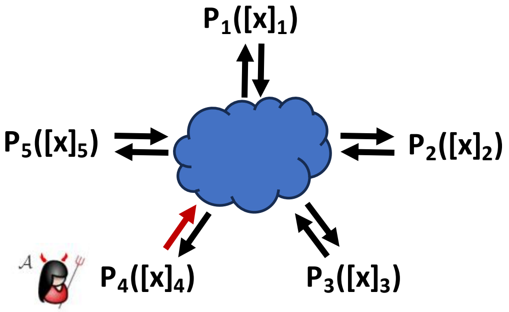
z <- P_i for $i \neq 4$

$\xrightarrow{\quad z \quad}$

Check consistency of all opened parties & verify result

Security - Soundness

Prover P (sk) | **Verifier V (pk)**



w <- Commit([x]), $\xrightarrow{\quad w \quad}$

$\xleftarrow{\quad c \neq 4 \quad}$ c $<-_R$ {1, … , 5}

z <- P_i forall i ≠ c $\xrightarrow{\quad z \quad}$

Check consistency of all opened parties & verify result

Security - Soundness

Soundness

- Only if $c = i_A$, A will go undetected!
- Soundness error = 1 / N for N parties

Special soundness:

- Valid openings for $c_1 \neq c_2$ reveal all P_i
- => Can recombine [x]

Security - HVZK

- Simulator samples random c first
- Generates P_i, $i \neq c$, honestly, with random inputs
- Choses communication of P_c such that result is correct
- Computes all other parts following protocol

Real life...

- We need the random e for multiplication check! (and for SDitH also the points r)
- Add a round trip ...

Commit

- Share secret [x], generate required number (say t) of multiplication triples $([a],[b],[c])_i$
- Commit to all the shares of one party together.
- Send commitments to V

Challenge 1

- Send t random values e_i for multiplication verification (SDitH: Also t random points r_i to evaluate polynomials on)

Response 1

- Run MPC protocol using commited shares and e_i
- Assemble and send communication of all multiplication verifications

Challenge 2

- Send random c within {1, … , N}

Response 2

- Send all shares of each party P_i, $i \neq c$

Verify

- Run MPC protocol with "opened parties" using communications of unopened party
- Check that all communications are consistent
- Check that final result is correct
  (usually, C is built such that result is 0)

Impact on security

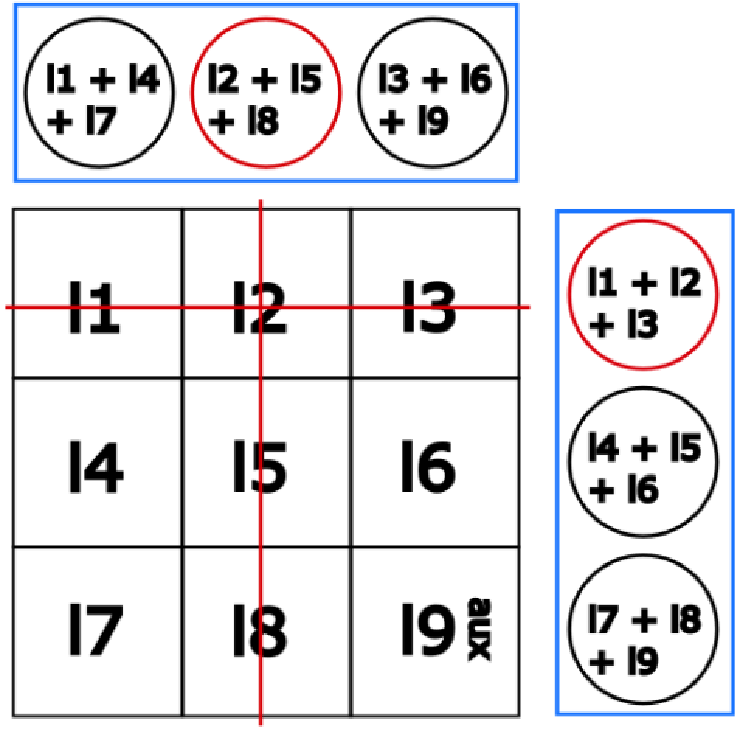
- HVZK: None – just sample all challenges in advance
- Soundness: Two ways of cheating -> guessing an e and manipulating the multiplication test or guessing the second challenge.
- Soundness error becomes $1 / |F_q| + 1 / N$

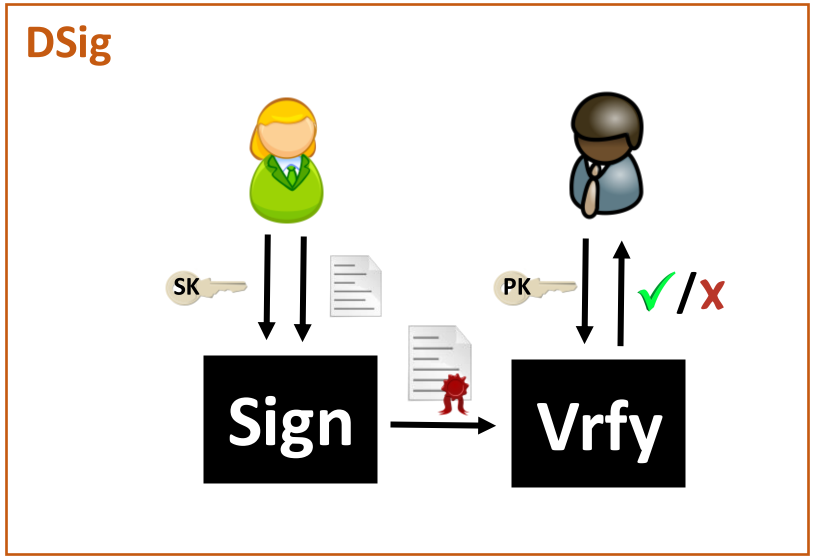
Optimizations

- Generate secret shares using PRG, e.g.:
  $x = [x]_1 + [x]_2 + \ldots + [x]_N + \Delta$ for $[x]_i = PRG(s_i)$ and $\Delta = x - \Sigma[x]_I$
  - requires to send the Δ in first communication!
  - Only need to commit to and later open s_i which are shorter than $[x]_i$
- Generate s_i using TreePRG
  - Allows to open all but one leaf publishing log N seeds in place of N!
- Hash commitment message and send unopened commitments in last message: $w' = H(w)$
  - Commitment-recoverable IDS
  - MUCH shorter w, only slightly longer z

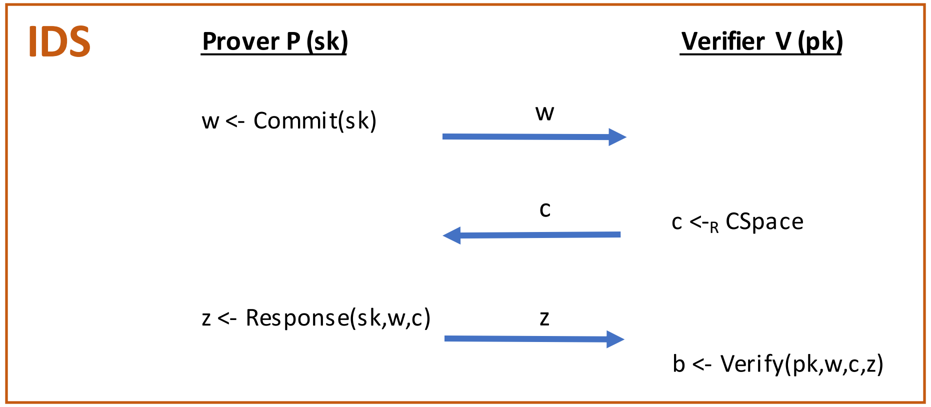
Hypercube verification

Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue
"The Return of the SDitH".
EUROCRYPT'23





Fiat-Shamir



Fiat-Shamir

Fiat-Shamir Signatures

Sign (sk,m)

1. w <- P.commit(sk)
2. c <- hash(pk, w, m)
3. z <- P.response(sk, w, c)
4. Return sig = (w, c, z)

Verify (pk, m, sig)

1. c <- hash(pk, w, m)
2. b <- V.verify(pk, w, c, z)

Why is this secure?

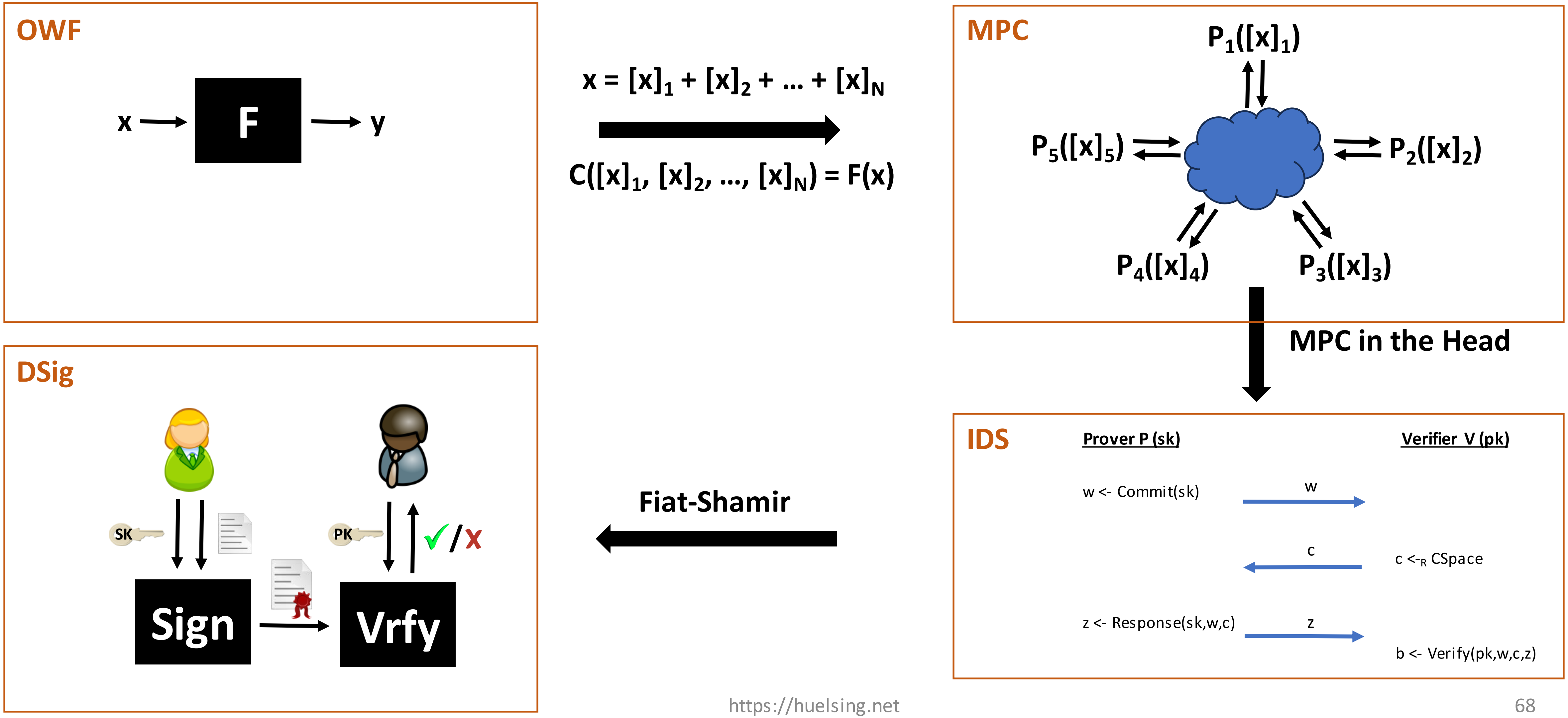
- HVZK -> Forger does not learn anything about the secret (or how to sign a different message) from seeing signatures on chosen messages
  - Proof idea: (Q)ROM proof.
  - Answer queries by running HVZK simulator
  - Program RO to make them consistent (set c <- H(w,m))
- Soundness -> Cannot do better than guessing the challenge per hash query / finding a suitable preimage for given challenge
  - For special case of 3-round commit & open IDS with special soundness doable in QROM, otherwise complicated (massive loss, hard proof)
    - If the adversary has higher success probability than the soundness error, it must be able to answer for more than one challenge.
    - All openings must be sound
    - Implementing the commitment using a random oracle, we can open all commitments using the random oracle table -> can generate two valid transcripts for different c & extract

SDitH in the QROM

(Aguilar-Melchor, Hülsing, Joseph, Majenz, Ronen, Yue. *SDitH in the QROM.* Asiacrypt'23)

- Can turn SDitH IDS into 3 round IDS replacing first challenge by hash of first message (FS but easier proof -> search problem)
- Get a scheme with query-bounded special soundness
- Apply FS for 3-round commit & open IDS in QROM

Summary



OWF
x
F
y
$x = [x]_1 + [x]_2 + \ldots + [x]_N$
$C([x]_1, [x]_2, \ldots, [x]_N) = F(x)$
MPC
$P_1([x]_1)$
$P_2([x]_2)$
$P_3([x]_3)$
$P_4([x]_4)$
$P_5([x]_5)$
MPC in the Head
IDS
Prover P (sk)
Verifier V (pk)
w <- Commit(sk)
w
c
$c \leftarrow_R$ CSpace
z <- Response(sk,w,c)
z
b <- Verify(pk,w,c,z)
Fiat-Shamir
DSig
SK
PK
✓/X
Sign
Vrfy

Conclusion

- MPCitH allows to build signature scheme from OWF
- Works best for functions with mostly linear steps
- Several nice optimizations exist
- Quite competitive:
  - small sk,
  - small pk,
  - medium sigs,
  - fast
  - allows for online / offline sigs

Table 1: Implementation benchmarks of Hypercube-SDitH vs our tweaked scheme for NIST security level I. For the PoW, the parameter $k_{iter} = D$ is used.

Scheme	Aim	Signature Size (bytes)	Parameters				Sign Time (in ms)			Verify Time
			$\lvert\mathbb{F}_{\text{points}}\rvert$	t	D	τ	Offline	Online	Total	(in ms) Total
Hypercube-SDitH [2]	Short	8464	2^{24}	5	8	17	3.83	0.68	4.51	4.16
	Shorter	6760	2^{24}	5	12	12	44.44	0.60	45.04	42.02
Ours Vanilla	Short	8464	2^{24}	5	8	17	4.45	0.049	4.50	4.17
	Shorter	6760	2^{24}	5	12	12	44.98	0.080	45.06	42.02
Ours PoW	Short	7968	2^{24}	5	8	16	4.20	0.14	4.34	4.00
	Shorter	6204	2^{24}	5	12	11	41.06	1.49	42.55	39.75

(Aguilar-Melchor, Hülsing, Joseph, Majenz, Ronen, Yue. *SDitH in the QROM.* Asiacrypt'23)