Internship Presentation

Mario Marhuenda Beltrán, Rafaël del Pino, Thomas Prest

September 6, 2022

1. Studied problem: Introduction to MPCiTH schemes

2. What did work

3. What did not work

Motivations for the problem

Signatures that only rely on symmetric crypto are more secure.
Currently there are only two serious families of schemes:

Motivations for the problem

Signatures that only rely on symmetric crypto are more secure.
Currently there are only two serious families of schemes:

- Hash-based signatures: Sphincs.
  - Size: From 8KB (slow) to 30KB (fast) (L1).

Motivations for the problem

Signatures that only rely on symmetric crypto are more secure.
Currently there are only two serious families of schemes:

- Hash-based signatures: Sphincs.
    - Size: From 8KB (slow) to 30KB (fast) (L1).
- MPC in the head (MPCitH) schemes.

Motivations for the problem

Signatures that only rely on symmetric crypto are more secure.
Currently there are only two serious families of schemes:

- Hash-based signatures: Sphincs.
  - Size: From 8KB (slow) to 30KB (fast) (L1).
- MPC in the head (MPCitH) schemes.

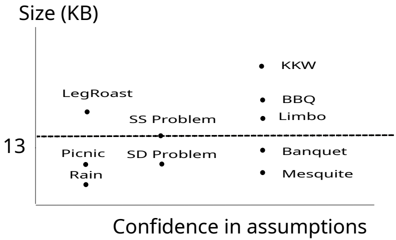

Figure: Comparison of the MPCiTH schemes

MPCitH: Original idea.

Input

- witness w
- Public key y
- Public function f
- Protocol Π that is t -private.

MPCitH: Original idea.

Input

- witness w
- Public key y
- Public function f
- Protocol Π that is t -private.

Construction of the signature:

$$y = f(x), \text{ f One Way Function (OWF)}$$

MPCitH: Original idea.

Input

- witness w
- Public key y
- Public function f
- Protocol Π that is t -private.

Construction of the signature:

$$y = f(x), \text{ f One Way Function (OWF)}$$

Honest Verifier Zero Knowledge Argument of Knowledge (HVZKAoK)

- **Completeness**: Accept if x is **known**.
- **Soundness**: Usually reject if x is **unknown**.
- **Zero knowledge**: Verifier learns **nothing** about x.

Description of IKOS

Prover	**Verifier**

Description of IKOS

Prover	**Verifier**
Simulates $\Pi_f(x, w_1, \ldots, w_n)$.	

Description of IKOS

Prover		**Verifier**
Simulates $\Pi_f(x, w_1, \ldots, w_n)$.		
	$\xrightarrow{(View_i)_{i=1}^n}$	

Description of IKOS

Prover		**Verifier**
Simulates $\Pi_f(x, w_1, \ldots, w_n)$.		
	$\xrightarrow{(View_i)_{i=1}^n}$	
		Chooses $T \subset [n]$, $\lvert T \rvert = t$, uniformly at random.

Description of IKOS

Prover		**Verifier**
Simulates $\Pi_f(x, w_1, \ldots, w_n)$.		
	$\xrightarrow{(View_i)_{i=1}^n}$	
		Chooses $T \subset [n]$, $\lvert T \rvert = t$, uniformly at random.
	$\xleftarrow{T}$	

Description of IKOS

Prover		**Verifier**
Simulates $\Pi_f(x, w_1, \ldots, w_n)$.		
	$\xrightarrow{(View_i)_{i=1}^n}$	
		Chooses $T \subset [n]$, $\|T\| = t$, uniformly at random.
	$\xleftarrow{T}$	
Opens the views of $\{P_i\}_{i \in T}$.		

Description of IKOS

Prover		**Verifier**
Simulates $\Pi_f(x, w_1, \ldots, w_n)$.		
	$\xrightarrow{(View_i)_{i=1}^n}$	
		Chooses $T \subset [n]$, $\lvert T \rvert = t$, uniformly at random.
	$\xleftarrow{T}$	
Opens the views of $\{P_i\}_{i \in T}$.		
	$\xrightarrow{(View_i)_{i \in T}}$	

Description of IKOS

Verifier's last step

Verifier accepts if all of the below conditions are met:

a. The prover succesfully opened the commits of the requested parties.

Description of IKOS

Verifier's last step

Verifier accepts if all of the below conditions are met:

a. The prover succesfully opened the commits of the requested parties.

b. The opened parties output x.

Description of IKOS

Verifier's last step

Verifier accepts if all of the below conditions are met:

a. The prover succesfully opened the commits of the requested parties.
b. The opened parties output x.
c. The opened parties' views are all consistent with each other.

Description of IKOS

Simple IKOS

Soundness error: $1 - t/n$. **Repeat** $O(n)$ times:

$$(1 - t/n)^{O(n)} = O(2^{-n})$$

Description of IKOS

Simple IKOS

Soundness error: $1 - t/n$. **Repeat** $O(n)$ times:

$$(1 - t/n)^{O(n)} = O(2^{-n})$$

Robust IKOS

The same algorithm can be used in the case where Π_f is t_p private and t_r robust, then the soundness is:

$$(1 - t_p/n)^{t_r}$$

In this case is possible to achieve negligible soundness in $O(1)$ rounds.

Removing interaction

Fiat-Shamir

- Signature schemes are non interactive!
- **Fiat-Shamir** transform.

Removing interaction

Fiat-Shamir

- Signature schemes are non interactive!
- **Fiat-Shamir** transform.

Signature from OWF

- Signature:
  - Public key: x.
  - Private key: y.
  - Signature: HVZKAoK that $y = f(x)$.

Removing interaction

Fiat-Shamir

- Signature schemes are non interactive!
- **Fiat-Shamir** transform.

Signature from OWF

- Signature:
  - Public key: x.
  - Private key: y.
  - Signature: HVZKAoK that $y = f(x)$.

Signature from AES

- Particular example:
  - Public key: (m, c).
  - Private key: k.
  - Signature: HVZKAoK that $c = AES_k(m)$.

Better protocols: Using correlated randomness.

- Goal: Use correlated randomness.
- `Beaver triples`
- `Summary`: Communication of 2 elements/mult gate.

Cut and choose: Keeping the prover honest

Prover	Verifier

Cut and choose: Keeping the prover honest

Prover	Verifier
Generates enough triples to run M executions of the protocols. And commits to them.	

Cut and choose: Keeping the prover honest

Prover		Verifier
Generates enough triples to run M executions of the protocols. And commits to them.		
	$\xrightarrow{\text{commit}}$	

Cut and choose: Keeping the prover honest

Prover		Verifier
Generates enough triples to run M executions of the protocols. And commits to them.		
	$\xrightarrow{\text{commit}}$	
		Asks to open $M - \tau$

Cut and choose: Keeping the prover honest

Prover		Verifier
Generates enough triples to run M executions of the protocols. And commits to them.		
	$\xrightarrow{\text{commit}}$	
		Asks to open $M - \tau$
	$M - \tau$	
	$\xrightarrow{\text{requested triples}}$	
		Verifies they are correct

Cut and choose: Keeping the prover honest

Prover		Verifier
Generates enough triples to run M executions of the protocols. And commits to them.		
	$\xrightarrow{\text{commit}}$	
		Asks to open $M - \tau$
	$M - \tau$	
	$\xrightarrow{\text{requested triples}}$	
		Verifies they are correct

What is the chance that a dishonest prover fools the verifier?

Cut and choose: Keeping the prover honest

Prover	Verifier

Cut and choose: Keeping the prover honest

Prover	Verifier

Runs preprocessing

Cut and choose: Keeping the prover honest

Prover		Verifier
	Runs preprocessing	
Runs τ execution		
	commit	
	$\xrightarrow{\text{views}}$	

Cut and choose: Keeping the prover honest

Prover		**Verifier**
	Runs preprocessing	
Runs τ execution		
	commit views $\longrightarrow$	
		Challenges every execution

Cut and choose: Keeping the prover honest

Prover		Verifier
	Runs preprocessing	
Runs τ execution		
	commit	
	$\xrightarrow{\text{views}}$	
		Challenges every execution
	$\xleftarrow{(\bar{i}_e)_{e\in\tau}}$	

Cut and choose: Keeping the prover honest

Prover		**Verifier**
	Runs preprocessing	
Runs τ execution		
	commit views $\longrightarrow$	
		Challenges every execution
	$\xleftarrow{(\bar{i}_e)_{e\in\tau}}$	
Opens all parties but $P_{\bar{i}_e\, e\in\tau}$		
	$\xrightarrow{Answers}$	

Cut and choose: Keeping the prover honest

Prover		**Verifier**
	Runs preprocessing	
Runs τ execution		
	commit views $\xrightarrow{}$	
		Challenges every execution
	$\xleftarrow{(\bar{i}_e)_{e\in\tau}}$	
Opens all parties but $P_{\bar{i}_e\, e\in\tau}$		
	$\xrightarrow{Answers}$	

Description KKW

Circuit size	1000 mult gates	10000 mult gates
	Signature size (KB)	Signature size (KB)
n = 64	37	136
n = 32	39	159
n = 16	44	190
n = 8	50	245

Another paradigm

One Beaver triple can verify another.

Another paradigm

One Beaver triple can verify another.

- Prover inserts the views of the parties, and then runs a protocol that checks that it was honest

Another paradigm

One Beaver triple can verify another.

- Prover inserts the views of the parties, and then runs a protocol that checks that it was honest
- Verifier issues a challenge, and prover runs check.

Circuit specific constructions

Adapt MPC protocol to particular OWF.

Circuit specific constructions

Adapt MPC protocol to particular OWF.

- **Before**:
  - Addition gates: Locally.
  - `Multiplication gates`: Two elements of communication.

Circuit specific constructions

Adapt MPC protocol to particular OWF.

- **Before**:
  - Addition gates: Locally.
  - `Multiplication gates`: Two elements of communication.
- **Now**:
  - Addition gates: Locally.
  - `'Inversion' gates`: At most three elements of communication.

BBQ and Banquet

- **BBQ**: Computation of the inverse gate. Parties share triples, and $r \in \mathbb{F}_{2^8} - \{0\}$.
  1. P_i has input x_i.
  2. The parties open $r \cdot x = (\sum r_i)(\sum x_i)$.
  3. P_i sets its output as: $r_i \cdot (r \cdot x)^{-1}$.

BBQ and Banquet

- **BBQ**: Computation of the inverse gate. Parties share triples, and $r \in \mathbb{F}_{2^8} - \{0\}$.
  1. P_i has input x_i.
  2. The parties open $r \cdot x = (\sum r_i)(\sum x_i)$.
  3. P_i sets its output as: $r_i \cdot (r \cdot x)^{-1}$.

- **Banquet**: Computation of the inverse gate. Suppose there are Ω gates:
  1. Prover shares the outputs.
  2. For the $k-th$ inversion gate, P_i sets $S^{(i)}(k-1) = s_k^{(i)}$.
  3. To preserve zk, they set $S^{(i)}(\Omega)$ and $T^{(i)}(\Omega)$ at random.
  4. Prover computes and shares: $P = S \cdot R$.
  5. Then the verifier chooses $\upsilon \xleftarrow{\$} \mathbb{F} - \{$ points already used for interpolation $\}$ and the parties open $P(\upsilon), R(\upsilon), S(\upsilon)$.
  6. Verifier checks that $P(\upsilon) \stackrel{?}{=} R(\upsilon) \cdot S(\upsilon)$

Custom ciphers

It is possible to construct ciphers that are MPCiTH friendly:

1. `LowMC`: Picnic scheme.

Custom ciphers

It is possible to construct ciphers that are MPCiTH friendly:

1. `LowMC`: Picnic scheme.
2. `Variants of Rijndael`: AES is a particular parameter selection of Rijndael, we can change the size of the S-boxes.

Custom ciphers

It is possible to construct ciphers that are MPCiTH friendly:

1. `LowMC`: Picnic scheme.
2. `Variants of Rijndael`: AES is a particular parameter selection of Rijndael, we can change the size of the S-boxes.
3. `LSAES`: Rainier scheme.
    - Simplified variant of Rijndael with a linearized key scheduled.

Custom ciphers

It is possible to construct ciphers that are MPCiTH friendly:

1. `LowMC`: Picnic scheme.
2. `Variants of Rijndael`: AES is a particular parameter selection of Rijndael, we can change the size of the S-boxes.
3. `LSAES`: Rainier scheme.
   - Simplified variant of Rijndael with a linearized key scheduled.
4. `Rain`: Rainier scheme.

Custom ciphers

It is possible to construct ciphers that are MPCiTH friendly:

1. `LowMC`: Picnic scheme.
2. `Variants of Rijndael`: AES is a particular parameter selection of Rijndael, we can change the size of the S-boxes.
3. `LSAES`: Rainier scheme.
    - Simplified variant of Rijndael with a linearized key scheduled.
4. `Rain`: Rainier scheme.
    - Cipher based on the Even-Mansour construction.
5. `LegRoast`: PRG using Legendre function.

Custom ciphers

It is possible to construct ciphers that are MPCiTH friendly:

1. `LowMC`: Picnic scheme.
2. `Variants of Rijndael`: AES is a particular parameter selection of Rijndael, we can change the size of the S-boxes.
3. `LSAES`: Rainier scheme.
   - Simplified variant of Rijndael with a linearized key scheduled.
4. `Rain`: Rainier scheme.
   - Cipher based on the Even-Mansour construction.
5. `LegRoast`: PRG using Legendre function.

Scheme	| pk| (bytes)	| sig | (bytes)	Sign	Verify
Banquet-AES-128	32	13284	47.31	43.03
Banquet-EM-AES-128	32	11940	41.05	36.88
Banquet-EM-LSAES-128	32	10496	20.99	18.91
Rainier-128	32	4880	28.28	28.16

Mesquite

Use a random polynomial of degree 2, $\mathcal{F}$, as a OWF.

$$G(x, y) = \mathcal{F}(x + y) - \mathcal{F}(x) - \mathcal{F}(y)$$

Mesquite

Use a random polynomial of degree 2, $\mathcal{F}$, as a OWF.

$$G(x, y) = \mathcal{F}(x + y) - \mathcal{F}(x) - \mathcal{F}(y)$$

Change of Beaver triples: (u,v) so that $v = \mathcal{F}$.

Mesquite

Use a random polynomial of degree 2, $\mathcal{F}$, as a OWF.

$$G(x, y) = \mathcal{F}(x + y) - \mathcal{F}(x) - \mathcal{F}(y)$$

Change of Beaver triples: (u,v) so that $v = \mathcal{F}$.

1. Step 1: Each P_i masks its inputs with u: $x_i - u_i$. Open $x - u = \sum_{i=1}^{n} x_i - u_i$.

Mesquite

Use a random polynomial of degree 2, $\mathcal{F}$, as a OWF.

$$G(x, y) = \mathcal{F}(x + y) - \mathcal{F}(x) - \mathcal{F}(y)$$

Change of Beaver triples: (u,v) so that $v = \mathcal{F}$.

1. Step 1: Each P_i masks its inputs with u: $x_i - u_i$. Open $x - u = \sum_{i=1}^{n} x_i - u_i$.
2. Step 2: Then P_i locally computes $o_i = G(u_i, x - u) + v_i$.

Mesquite

Use a random polynomial of degree 2, $\mathcal{F}$, as a OWF.

$$G(x, y) = \mathcal{F}(x + y) - \mathcal{F}(x) - \mathcal{F}(y)$$

Change of Beaver triples: (u,v) so that $v = \mathcal{F}$.

1. Step 1: Each P_i masks its inputs with u: $x_i - u_i$. Open $x - u = \sum_{i=1}^{n} x_i - u_i$.
2. Step 2: Then P_i locally computes $o_i = G(u_i, x - u) + v_i$.
3. Step 3: Parties open

$$G(u, x - u) + F(u) = F(x) - F(x - u)$$

New stuff

Improvements on Mesquite There are number of improvements presented in Banquet that are directly applicable to Mesquite

New stuff

Improvements on Mesquite There are number of improvements presented in Banquet that are directly applicable to Mesquite

1. `Removing the output broadcast:`

New stuff

Improvements on Mesquite There are number of improvements presented in Banquet that are directly applicable to Mesquite

1. `Removing the output broadcast:`
2. `Better commitments`

New stuff

Improvements on Mesquite There are number of improvements presented in Banquet that are directly applicable to Mesquite

1. `Removing the output broadcast:`
2. `Better commitments`
3. `Amortizing party seeds.`

New stuff

Improvements on Mesquite There are number of improvements presented in Banquet that are directly applicable to Mesquite

1. `Removing the output broadcast:`
2. `Better commitments`
3. `Amortizing party seeds.`

Mesquite formula

$$2\kappa + 3\kappa\tau \left\lceil \log \frac{M}{\tau} \right\rceil + \tau \left(\kappa \lceil \log N \rceil + \kappa + (n+m) \log q \right)$$

New stuff

Improvements on Mesquite There are number of improvements presented in Banquet that are directly applicable to Mesquite

1. `Removing the output broadcast:`
2. `Better commitments`
3. `Amortizing party seeds.`

Mesquite formula

$$2\kappa + 3\kappa\tau \left\lceil \log \frac{M}{\tau} \right\rceil + \tau\left(\kappa \lceil \log N \rceil + \kappa + (n+m)\log q\right)$$

New formula

$$4\kappa + 2\kappa\tau \left\lceil \log \frac{M}{\tau} \right\rceil + \tau\left(\kappa \lceil \log N \rceil + \kappa + n\log q\right)$$

Formula comparison

Improvements on Merkle tree

			Tree cost (KB)	
N	M	τ	Before	After
8	176	51	4.34	2.31
16	232	37	4.625	2.42

Formula comparison

Improvements on Merkle tree

			Tree cost (KB)	
N	M	τ	Before	After
8	176	51	4.34	2.31
16	232	37	4.625	2.42

Improvements on general scheme

			Sig size (KB)	
N	M	τ	Mesquite	Updated
8	176	51	10.51	7.42
16	232	37	9.68	6.77

Multivariate sacrificing

Prove: Prove $F(x) = y$, $y \in \mathbb{F}_q^M$ public $x \in \mathbb{F}_q^N$ secret.

Multivariate sacrificing

Prove: Prove $F(x) = y$, $y \in \mathbb{F}_q^M$ public $x \in \mathbb{F}_q^N$ secret.

It turns out we can adapt Banquet's proof.

Multivariate sacrificing

Prove: Prove $F(x) = y$, $y \in \mathbb{F}_q^M$ public $x \in \mathbb{F}_q^N$ secret.

It turns out we can adapt Banquet's proof.

Soundness: Can be bounded by Schwartz-Zippel lemma $\frac{2}{q^n - 3}$.

Multivariate sacrificing

Prove: Prove $F(x) = y$, $y \in \mathbb{F}_q^M$ public $x \in \mathbb{F}_q^N$ secret.

It turns out we can adapt Banquet's proof.

Soundness: Can be bounded by Schwartz-Zippel lemma $\frac{2}{q^n-3}$.

Compute soundness for parameters

- $P_1(\tau, \tau_1) = PMF(B(\tau, \tau_1, p)) = \sum_{k=\tau_1}^{\tau} \binom{\tau}{k} p^k (1-p)^{\tau-k}$

Multivariate sacrificing

Prove: Prove $F(x) = y$, $y \in \mathbb{F}_q^M$ public $x \in \mathbb{F}_q^N$ secret.

It turns out we can adapt Banquet's proof.

Soundness: Can be bounded by Schwartz-Zippel lemma $\frac{2}{q^n - 3}$.

Compute soundness for parameters

- $P_1(\tau, \tau_1) = PMF(B(\tau, \tau_1, p)) = \sum_{k=\tau_1}^{\tau} \binom{\tau}{k} p^k (1-p)^{\tau-k}$
- $P_2(\tau, \tau_1) = N^{\tau - \tau_1}$

Multivariate sacrificing

Prove: Prove $F(x) = y$, $y \in \mathbb{F}_q^M$ public $x \in \mathbb{F}_q^N$ secret.

It turns out we can adapt Banquet's proof.

Soundness: Can be bounded by Schwartz-Zippel lemma $\frac{2}{q^n - 3}$.

Compute soundness for parameters

- $P_1(\tau, \tau_1) = PMF(B(\tau, \tau_1, p)) = \sum_{k=\tau_1}^{\tau} \binom{\tau}{k} p^k (1-p)^{\tau-k}$
- $P_2(\tau, \tau_1) = N^{\tau - \tau_1}$
- $\min 1/P_1 + 1/P_2$

Choosing the right parameters

Analysis soundness. We want to find:

1. N,M: $\mathcal{F} : \mathbb{F}_q^N \to \mathbb{F}_q^M$
2. τ
3. n

Things that did not work

Size formula:
$6\kappa + \tau\kappa \cdot \lceil \log N \rceil + \tau \cdot (2\kappa + (2m + n) \log q)$

Things that did not work

Size formula:
$6\kappa + \tau\kappa \cdot \lceil \log N \rceil + \tau \cdot (2\kappa + (2m+n)\log q)$

N	τ	τ_1	τ_2	security level	signature size
8	43	0	32	128.0	6.45KB
16	32	0	26	128.0	5.31KB
32	26	0	26	130.0	4.73KB
64	22	0	22	132.0	4.36KB
128	19	0	19	133.0	4.07KB
1024	13	0	13	130.0	3.41KB
65536	8	0	8	128.0	2.8KB

Things that did not work

Size formula:
$6\kappa + \tau\kappa \cdot \lceil \log N \rceil + \tau \cdot (2\kappa + (2m + n) \log q)$

N	τ	τ_1	τ_2	security level	signature size
8	43	0	32	128.0	6.45KB
16	32	0	26	128.0	5.31KB
32	26	0	26	130.0	4.73KB
64	22	0	22	132.0	4.36KB
128	19	0	19	133.0	4.07KB
1024	13	0	13	130.0	3.41KB
65536	8	0	8	128.0	2.8KB

Asymptotic limit: 2.2KB.

Things that did not work

Can we actually use robust protocols?

Things that did not work

Can we actually use robust protocols?

In particular, we take:

$$\mathcal{C} = \{(p(1), \ldots, p(n) : p \in \mathbb{F}_p[x], \deg(p) < k\}$$

Things that did not work

Can we actually use robust protocols?

In particular, we take:

$$\mathcal{C} = \{(p(1), \ldots, p(n) : p \in \mathbb{F}_p[x], \deg(p) < k\}$$

Soundness error

$$\left(1 - \frac{k}{n}\right)^{d-1}$$

Other things that did not work

1. Smart ways compute the S-box.

Other things that did not work

1. Smart ways compute the S-box.
   - Masked tables.
   - Special gates.

Other things that did not work

1. Smart ways compute the S-box.
    - Masked tables.
    - Special gates.
    - Homomorphic hashing: Relies on CVP.
    - Vector commitments?

Other things that did not work

1. Smart ways compute the S-box.
    - Masked tables.
    - Special gates.
    - Homomorphic hashing: Relies on CVP.
    - Vector commitments?
2. Amortizing repetition.

Other things that did not work

1. Smart ways compute the S-box.
    - Masked tables.
    - Special gates.
    - Homomorphic hashing: Relies on CVP.
    - Vector commitments?
2. Amortizing repetition.
3. VSS

Other things that did not work

1. Smart ways compute the S-box.
   - Masked tables.
   - Special gates.
   - Homomorphic hashing: Relies on CVP.
   - Vector commitments?
2. Amortizing repetition.
3. VSS
4. Other ‘standard’ symmetric ciphers.

Summary and future work

Summary

1. MPCiTH
2. Multivariate sacrificing

Summary and future work

Summary

1. MPCiTH
2. Multivariate sacrificing

Future work

1. Implementation of Multivariate Sacrificing (Cranberry). (How high can n be?).
2. Proof of security.
3. Rescue some ideas.