Presentation Monday

Mario

June 16, 2022

Outline

1. MPCitH
2. KKW and BN proofs.
3. Optimizations that are already known.
4. **TODO** Other MPCitH protocols.
5. Unexplored options
6. Alternatives.

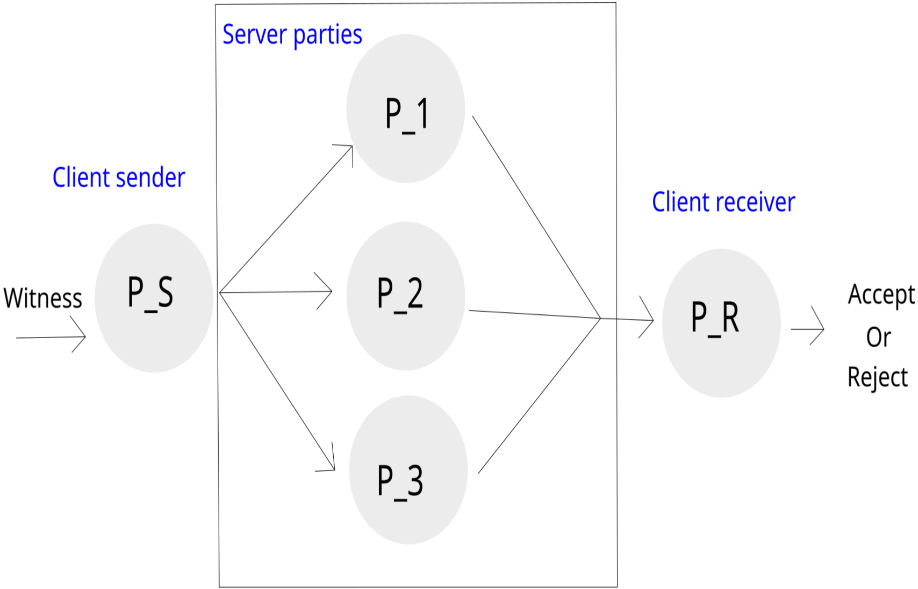
MPCitH: Original idea.

Input

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

1. Choose $w_i \overset{unif}{\leftarrow} \{0,1\}^m$ so that $\sum w_i = w$.
2. Prover runs the protocol $\Pi(f, x, w_1, \ldots, w_n)$.
3. And commits to the "views of the parties".
   - $\text{View}_i = w_i, m_{i,j}, r_i$. ($m_{i,j}$ = messages from P_i to P_j).
   - r_i randomness of P_i.
4. Verifier chooses $T \subset \{1, \ldots, n\}$, $|T| = t$.
5. Prover opens the views of parties $t \in T$.
6. Verifier checks that the opened views are consistent with their commits and that they output correct.

Ligero(++)'s and Limbo's choice



Better solutions: Using correlated randomness.

- Goal: Use correlated randomness.
- Wishlist: Better MPC protocols compared to BGW.
  - SPDZ
  - `Beaver triples`: Objective: Known (x_i, y_i) find (z_i) so that: $\sum z_i = (\sum x_i)(\sum y_i)$, without sharing x or y. Given triples (a, b, c) and a sharing (a_i, b_i, c_i) then, choose ε at random and do:
    - Compute $\alpha_i = x_i - a_i$ and $\beta_i = y_i - b_i$
    - Open α and β, i.e $\alpha = \sum \alpha_i \; \beta = \sum \beta_i$
    - Compute: $z_i = c_i - \alpha \cdot b - \beta \cdot a_i + \alpha \cdot \beta$

KKW

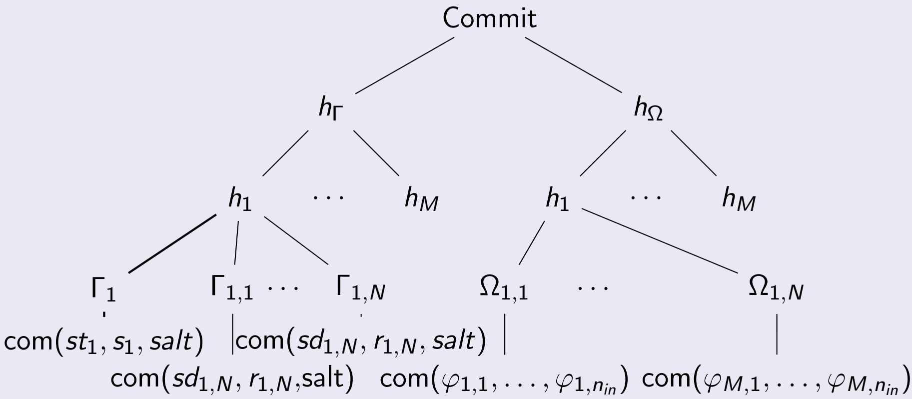
Idea: Randomness is verified for some parties before the execution of Π_f, and the verifier trusts that if the opened parties are honest, so are the unopened ones.

1. Preprocessing phase
2. Generate $M \cdot n_{\text{mult}}$ random triples $(x_i, y_i, z_i)_{i=0}^{N}$.
3. Verifier asks to see the randomness assigned to $M - \tau$ parties.
4. Verifies that the triples are correct.
5. Execute IKOS algorithm on τ unopened parties.

- Parameters:
  1. N: Number of parties.
  2. M: Number of preprocessing rounds.
  3. τ cut parameter: How many rounds of protocol will be executed in the end.
- An obvious optimization:
  1. Generate random triples from seed sd, and correct z_n to correct value: The prover need only give the verifier sd_i, which is the seed used by party i. It can be generated by $sd_i = H(sd||i)$.

KKW A better picture. Rounds 1,2.

Round 1: Commit of the randomness and input



Round 2: Commit of the randomness and input

Verifier chooses $E \subset [M]$, $|E| = \tau$ as a challenge.

KKW A better picture. Rounds 3.

Round 3: Prover runs the circuit. Online phase

- Prover generates views for every party, $view_e$ in the following way:
- $view_e \leftarrow$ ""
- For every mult gate (or square gate), call it g_k:
  - $view_e \leftarrow view_e || \alpha_{e,k,1} || \cdots \alpha_{e,k,N} || \beta_{e,k,1} || \cdots \beta_{e,k,N}$
- Add the outputs to the views, if k is the k output wire:
- $view_e \leftarrow view_e || o_{e,k,1} || \cdots o_{e,k,N}$
- Each party commits to their views: $\Pi_e = com(view_e, g_e, salt)$ (g_e freshly randomly generated).
- Hash everything $h_\pi = H(\Pi_e)_{e=1}^{|M-E|}$ and send to the verifier.

KKW A better picture. Rounds 4,5.

Round 4: Prover runs the circuit. Online phase

- For every $e \in M - E$ the verifier issues a challenge $i_e \in [N]$.

Round 5: Prover accepts the challenge

Prover sends the following things:

- Salt used to commit
- Fresh randomness used in Round 3.
- For the parties challenged (to be opened): Their seeds
- The commits from the unopened parties necessary to reconstruct h_Γ, h_Ω and h_π.
  This includes the α, β and correction bits of the unopened parties.

Finally the verifier recomputes the hashes with the given information. If the hashes agree, the prover succeedes, otherwise the prover fails.

KKW Part 2

- Soundness (ζ_{cc}):

$$\max_{\{0 \leq \tau < M\}} \max_{\{0 \leq c \leq M-\tau\}} \left\{ \frac{\binom{M-c}{\tau}}{\binom{M}{\tau} N^{M-\tau-c}} \right\}$$

- Communication complexity:
$2|hash| + \tau|sd| + |hash| + (M-\tau)(|com|) + 3\log_2|\mathbb{F}| +$
$|sd| + (M-\tau)(logN|sd|) + (M-\tau)(|com| +$
$(M-\tau)\log_2|\mathbb{F}|(3n_{mult} + 2n_{sq} + n_{in} + 1)$

BN

Idea: Instead of proving correct execution of circuit C, $\mathcal{P}$ simulates checking that the gates were computed correctly. He does it using the following:

- It is possible for the verifier to check that a multiplication has been done correctly by inserting some randomness.

  $\sum z_i = (\sum x_i)(\sum y_i)$, without sharing x or y.
  Given triples (a, b, c) and a sharing (a_i, b_i, c_i) then choose ε at random and do:
  1. Compute $\alpha_i = \varepsilon x_i + a_i$ and $\beta_i = y_i + b_i$
  2. Open α and β, i.e
  $\alpha = \sum \alpha_i$, $\beta = \sum \beta_i$
  3. Compute:
  $v_i = \varepsilon z_i - c_i - \alpha \cdot b_i + \beta \cdot a_i - \alpha \cdot \beta$
  Then $v = \sum v_i = \varepsilon \Delta_z - \Delta_c$
  $\Delta_z = \sum z_i - \sum x_i \sum y_i$, $\Delta_c = \sum c_i - \sum a_i \sum b_i$
  4. Check that $v = 0$.

BN Part 2

- BN algorithm:
  1. Imperfect preprocessing:
     1. Generate random triples. No inmediate challenge.
     2. Run the circuit and save the output value of each gate.
     3. Commit to the "state". That is: The seed used for each party, the correction bits and the output of each gate. Important: Prover sends one commit string overall. We call this commit h_Γ.
  2. Challenge: The verifier chooses a random seed, used to generate the challenges.
  3. Prover computes challenges on all gates, commits to the result of each party and sends it to the verifier. We call this commit h_π.
  4. Verifier chooses to all-but-one party per round.
  5. Prover sends the requested parties plus the complementary commits necessary to compute the hash tree.
  6. Verifier uses the data given by the verifier to compute the global hashes
- Parameters:
  1. N: Number of parties.
  2. M: Number of preprocessing rounds.

Sacrificing algorithm. Round 1,

Round 1

- Prover chooses some randomness: $salt$, sd_e and $sd_{e,i}$. (Salt for the commits, master seed per round and master seed per party and round.
- Prover generates empty strings st_e and $st_{e,i}$.
- Prover generates correction elements for the Beaver triples: $\Delta_{e,k}$ (Per party and per mult gate). And appends $\Delta_{e,k}$ to st_e.
- Prover runs the circuit. Generates correction elements for the mult gates:
  - Let g_k be a mult gate. With inputs $([[x_k]], [[y_k]])$.
  - Prover generates $([[z_k]])$ at random, computes (locally) $\varphi_{e,k} = x_k \cdot y_k - z_k$ which is the correction necessary to turn $([[z_k]])$ into a correct sharing of the output of the gate. And appends $\varphi_{e,k}$ to st_e.
- Prover commits to the state, same as KKW. (Now the states are different!).

Sacrificing algorithm. Round 2,3

Round 2

- Verifier chooses sd_θ and sends it to the Prover.

Round 3

- Prover generates views for every party, $view_e$ in the following way:
- $view_e \leftarrow$ ""
- For every mult gate (or square gate), call it g_k:
  - Use the verifier seed to generate $\epsilon_{e,k,i}$ and use them to challenge the multiplication: $view_e \leftarrow view_e || \alpha_{e,k,1} || \cdots \alpha_{e,k,N} || \beta_{e,k,1} || \cdots \beta_{e,k,N}$
- Add the $v_{e,k,i}$ to the view.
- Add the sharing of the outputs to the wires to the view.
- Send $h\pi$ to the verifier. (Defined as before, hash of the commits of the views).

Sacrificing algorithm. Round 4.5

Round 4

Verifier asks to open all-but-one party for every round.

Round 5

Prover sends enough information to allow the verifier to reconstruct the hashes.
In particular: Sends the seeds of the opened parties, and the α,β and v of the unopened party.

BN The global picture

$\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$:

Phase 1: Committing to the seeds and views of the parties.

Sample a random salt $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$.
for each parallel repetition e **do**
  Sample a root seed: $\mathsf{seed}_e \xleftarrow{\$} \{0,1\}^{\kappa}$.
  Derive $\mathsf{seed}_e^{(1)}, \ldots, \mathsf{seed}_e^{(N)}$ as leaves of binary tree from seed_e.
  for each party i **do**
    Commit to seed: $\mathsf{com}_e^{(i)} \leftarrow \mathsf{Commit}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$.
    Expand random tape: $\mathsf{tape}_e^{(i)} \leftarrow \mathsf{ExpandTape}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$
    Sample witness share: $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
  Compute witness offset: $\Delta\mathsf{sk}_e \leftarrow \mathsf{sk} - \sum_i \mathsf{sk}_e^{(i)}$.
  Adjust first share: $\mathsf{sk}_e^{(1)} \leftarrow \mathsf{sk}_e^{(1)} + \Delta\mathsf{sk}_e$.
  for each multiplication gate with index $\ell \in [C]$ **do**
    For each party i, set $(a_{e,\ell}^{(i)}, b_{e,\ell}^{(i)}, c_{e,\ell}^{(i)}) \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
    Compute $a_{e,\ell} = \sum_{i=1}^{N} a_{e,\ell}^{(i)}$, $b_{e,\ell} = \sum_{i=1}^{N} b_{e,\ell}^{(i)}$, $c_{e,\ell} = \sum_{i=1}^{N} c_{e,\ell}^{(i)}$.
    Compute offset $\Delta c_{e,\ell} = a_{e,\ell} \cdot b_{e,\ell} - c_{e,\ell}$.
    Adjust first share: $c_{e,\ell}^{(1)} \leftarrow c_{e,\ell}^{(1)} + \Delta c_{e,\ell}$
  for each gate g in $\mathcal{C}$ with index ℓ **do**
    if g is an addition gate with inputs (x, y) **then**
      The parties locally compute the output share:
      $z^{(i)} = x^{(i)} + y^{(i)}$
    if g is a multiplication gate with inputs $(x_{e,\ell}, y_{e,\ell})$ **then**
      Compute output shares $z_{e,\ell}^{(i)} = \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
      Compute offset $\Delta z_{e,\ell} = x_{e,\ell} \cdot y_{e,\ell} - \sum_{i=1}^{N} z_{e,\ell}^{(i)}$.
      Adjust first share $z_{e,\ell}^{(1)} \leftarrow z_{e,\ell}^{(1)} + \Delta z_{e,\ell}$.

BN The global picture

Let $\mathsf{ct}_e^{(i)}$ be the output shares of online simulation.

Set σ_1 to:

$(\mathsf{salt}, ((\mathsf{com}_e^{(i)})_{i\in[N]}, (\mathsf{ct}_e^{(i)})_{i\in[N]}, \Delta\mathsf{sk}_e, (\Delta c_{e,\ell}, \Delta z_{e,\ell})_{\ell\in[C]})_{e\in[\tau]}$.

Phase 2: Challenging the checking protocol.

Compute challenge hash: $h_1 \leftarrow H_1(\mathsf{salt}, \mathsf{msg}, \sigma_1)$.

Expand hash: $((\epsilon_{e,\ell})_{\ell\in[C]})_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_1)$ where $\epsilon_{e,\ell} \in \mathbb{F}$.

Phase 3: Commit to simulation of checking protocol.

for each multiplication gate with index $\ell \in [C]$ **do**

Simulate the triple checking protocol as defined above. Let $\alpha_{e,\ell}^{(i)}$ and $\beta_{e,\ell}^{(i)}$ be the two broadcast values and let $v_{e,\ell}^{(i)}$ be the output of the checking protocol, for all parties $i \in [N]$.

Set $\sigma_2 \leftarrow (\mathsf{salt}, (((\alpha_{e,\ell}^{(i)}, \beta_{e,\ell}^{(i)}, v_{e,\ell}^{(i)})_{i\in[N]})_{\ell\in[C]})_{e\in[\tau]}$.

Phase 4: Challenging the views of the MPC protocol.

Compute challenge hash: $h_2 \leftarrow H_2(h_1, \sigma_2)$.

Expand hash: $(\bar{i}_e)_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_2)$ where $\bar{i}_e \in [N]$.

Phase 5: Opening the views of the checking protocol.

for each repetition e **do**

$\mathsf{seeds}_e \leftarrow \{\log_2(N)$ nodes to compute $\mathsf{seed}_{e,i}$ for $i \in [N] \setminus \{\bar{i}_e\}\}$.

Output $\sigma \leftarrow (\mathsf{salt}, h_1, h_2, (\mathsf{seeds}_e, \mathsf{com}_e^{(\bar{i}_e)}, \Delta\mathsf{sk}_e, \mathsf{ct}_e^{(\bar{i}_e)}, (\Delta c_{e,\ell}, \Delta z_{e,\ell}, \alpha_{e,\ell}^{(\bar{i}_e)}, \beta_{e,\ell}^{(\bar{i}_e)}, v_{e,\ell}^{(\bar{i}_e)})_{\ell\in[C]})_{e\in[\tau]})$.

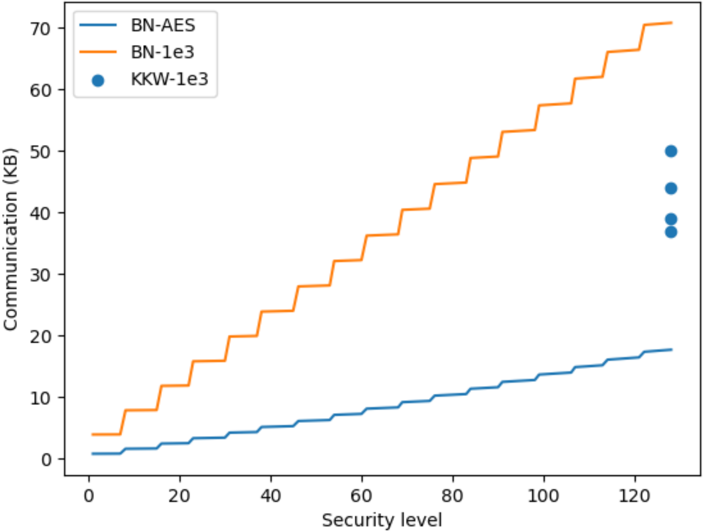
BN soundness and comunication

- Soundness:

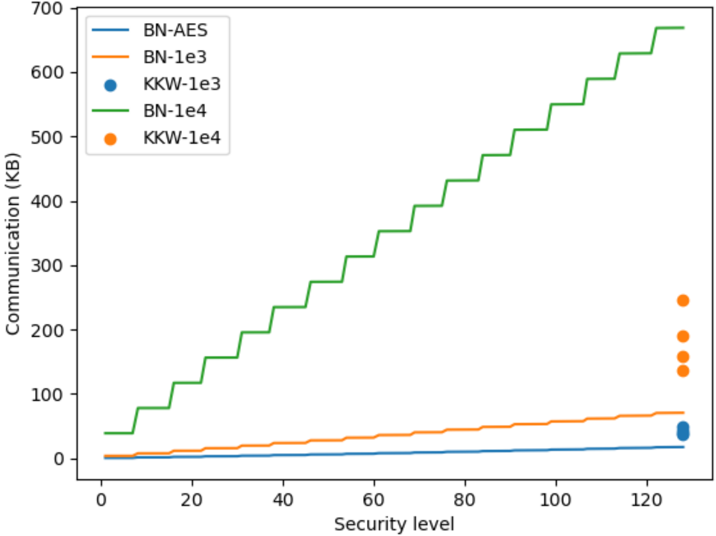
$$\left(\frac{2N+|\mathbb{F}|-2}{|\mathbb{F}|N}\right)^{M}$$

- Communication complexity: $\mid \text{hash} \mid + \mid \text{hash} \mid + \mid \text{sd} \mid +$
$M(\mid \text{sd} \mid + \mid \text{com} \mid + 4\log_2(|\text{F}|)$
$n_{\text{mult}} + 3\log_2(|\text{F}|)\text{n}_{\text{sq}} + \log_2(|\text{F}|)\text{n}_{\text{in}} + 2\log_2(|\text{F}|)$ $)$

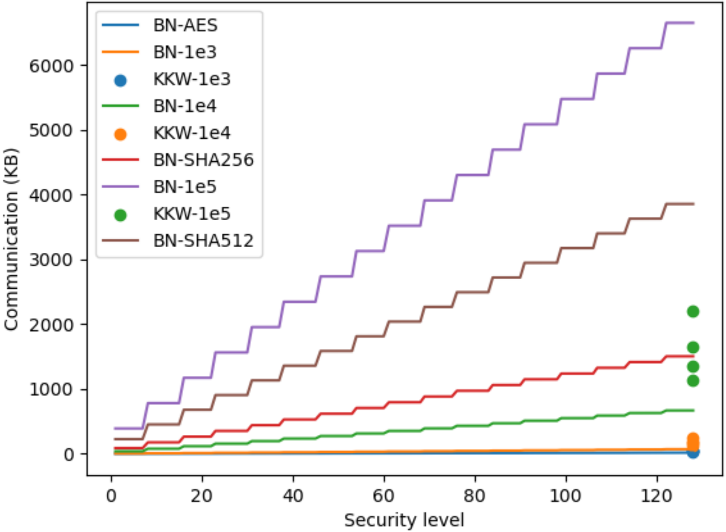
Some graphical comparisons



Some graphical comparisons.



Some graphical comparisons.



BN Generalization: Samping on the fly.

It is possible to formulate a general framework for the "sacrificing scheme".

1. Per round $e \in [M]$: $\mathcal{P}$ generates an extended witness $\hat{w}_e$.
2. Per $e \in [M]$, $\mathcal{V}$ samples $\tau_{\mathcal{V},e}$ and sends $(\tau_{\mathcal{V},e})_{e=1}^{M}$ to $\mathcal{P}$
3. Per $e \in [M]$, $\mathcal{P}$ computes $\tau_{\mathcal{P},e}$ and uses $\hat{w}$, $\tau_{\mathcal{V},e}$ and $\tau_{\mathcal{P},e}$ to sample a circuit C_e.
4. Run regular protocol with circuit C_e.

Output: $\mathcal{V}$ recomputes the circuit C_e and checks that $\mathcal{P}$'s transcripts are consistent.

Sampling on the fly: Examples

1. Supposse $w \in \mathbb{Z}^n$ was binary, we can extend $\hat{w} = w || w_1^2 - w_1 || \cdots || w_n^2 - w_n$.
2. If C has output $o_1, \ldots, o_n$, C_e can be set the same as C but output $\sum \gamma_i o_i$ with γ_i sampled from seed given by $\mathcal{V}$.

KKW zk theorem.

Theorem

KKW works Suppose that:

1. *H is a CRHF.*
2. *com is a Random Oracle-based commitment scheme.*

Then, KKW (i.e cut and choose) is HVZKAoK.

KKW proof.

Proof.

Let $\mathcal{S}_\Pi$ be a simulator for the MPC protocol. We want to build another simulator $\mathcal{S}$.

1. First $\mathcal{S}$ chooses random $E \subset [M]$. And random $i_e, \forall e \in [M] - E$.
   Except: $\mathcal{S}$ commits to a 0 string.
   - E will be the set of opened rounds of preprocessing.
   - i_e will be the unopened party in the unopened rounds of preprocessing.
2. $\forall e \in E$, the simulator computes the pp honestly, meaning:
   - Beaver triples are correct.
   - Share of the inputs look unif random.

□

KKW proof.

Proof.

1. $\forall e \in [M] - E$, the simulator cheats:
   - $\mathcal{S}$ generates seeds $sd_{e,i}$ but the correction bits are random.
   - $\mathcal{S}$ chooses random input shares. (I.e the shares do not reconstruct a valid witness).
   - $\mathcal{S}$ commits to 0-strings.
   - i_e will be the unopened party in the unopened rounds of preprocessing.
   - The views of i_e are "prepared" so that the protocol outputs accept.
2. $\mathcal{S}$ runs the protocol.
3. $\mathcal{S}$ computes the hash values of the commitments.
4. $\mathcal{S}$ outputs the transcript.

Then, if the transcript of $\mathcal{S}$ can be distinguished from the one of an honest prover, it implies that $\mathcal{S}_{\Pi}$ is not secure. □

KKW soundness.

Theorem

Suppose that:

1. *H is a CRHF.*
2. *com is a Random Oracle-based commitment scheme.*

Let $\delta(x)$ be the cheating probability, if it's larger than

$$\max_{0 \leq \tau \leq M} \max_{0 \leq c \leq M-\tau} \left\{ \frac{\binom{M-c}{\tau}}{\binom{M}{\tau} N^{M-\tau-c}} \right\}$$

Then the prover commited to a valid witness.

KKW soundness proof.

Proof.

First theorem The idea is to consider a 0/1 matrix G, so that the columns corresponds to the first challenge and the rows to the second. And 0 indicate failure and 1 success (either cheating or honest).
Then $\delta = \frac{|1s|}{\binom{M}{\tau} N^{M-\tau}}$
Using the hypothesis, we can deduce (by the pidgeonhole principle) that there must be two different valid executions, that have the same first challenge, but the second is different.
This would allow an extractor to produce the witness. □

KKW soundness proof 2.

Proof.

Second theorem Let $\varepsilon \in (0,1)$ so that: $\delta(x) = \zeta_{cc}(M, N, \tau) + \varepsilon$

Then the extractor, Υ, proceeds as follows:

1. Find some 1 entry in G. This entry is associated with some challenge: $(c_1, \ldots, c_n)$ (here c_i is the opened party in *ith* execution, so we can say $c_i = 0$ if the randomness is opened.
2. For each round execution, i, we run another extractor Υ_i, which finds a 1 entry in G which has a different challenge c_e' different from c_e.
3. For each c_e' outputted in the previous step, we extract a witness w, which we can do by the previous theorem.
4. Chech if for some of the previous witnesses, $C(w) = y$.

This succeds with prob $\geq \varepsilon/M$. So the algorithm runs in $O(M/\varepsilon)$.

□

Sacrificing zk theorem.

Theorem

KKW works Suppose that:

1. *H is a CRHF.*
2. *com is a Random Oracle-based commitment scheme.*

Then, Sacrificing is HVZKAoK.

Sacrificing proof

Proof.

A simulator follows the protocol honestly, except for the unponened party is given "prepared views". □

Better solutions for AES.

- `Preprocessing`: BBQ, Rainier
  - BBQ: Masked inversions
    - It is possible to treat the S-box as one gate: Let $(s_i) \in \mathbb{F}_{2^8}$ and random $(r_i) \in \mathbb{F}_{2^8}$:
    - Use the Beaver triples technique to compute a sharing of $s \cdot r$ and open it.
    - Compute $s^{-1} \cdot r^{-1}$:
    - Almost negligible effect on Key size (costs about 2-4 bits of security).
    - The parties compute their share of s^{-1} by: $s^{-1} \cdot r^{-1} \cdot r_i$.
    - `Communication`: 4 bytes elements per S-box.
  - Rain: AES with larger S-box to reduce the number of rounds.
    - `Requires symmetric crypto Budu`.

Dealing with 0

- What if $s \cdot r = 0$?

  1. Compute S-box by square and multiply: 33 bytes per S-box.
  2. Actually 19 bytes using the linearity of $f(x) = x^{2^i}$.
  3. Compute: $(s + \delta(s))^{-1} - 1_{\{0\}}(s)$ instead. 2 methods:
     1. Implementing AES as a binary circuit: 6.6 bytes
     2. Compute $\delta(s)$ as a n-party functionality.
        1. Prover shares $\delta(s)$ and gives correction bit.
        2. `Analysis is kept for further work!`.

Sacrificing: Banquet

- Same idea as in BBQ: The S-box can be verified like it is a single gate:
  - Let m be the number of S-boxes.
  - Let $s_1^{(i)}, \ldots, s_m^{(i)}$ be the outup of the S-boxes and $t_1^{(i)}, \ldots, t_m^{(i)}$ be sharing of the inverses.
  - Goal: Verify that,

$$\left(\sum_j s_j^{(i)}\right) \cdot \left(\sum_j t_j^{(i)}\right) = 1, \quad \forall j \in [m]$$

Sacrificing: Banquet. Naive idea.

- We want to use Schwartz-Zippel.
- Verifier isssues random challenge.
- Let m be the number of S-boxes, (i.e 20?/party).
  - s_i is the input of the i th S-box and t_i is it's output.
  - Locally each party computes $S(\cdot), T(\cdot)$ so that $S(i) = s_i, T(i) = i$ and one last random point.
  - We define $P = ST$.
  - Then parties locally compute $P(R) = S(R)T(R)$.
  - Verifier reconstruct P (using $2m + 1$ points, but $m + 1$ are already given).

Sacrificing: Banquet. Better Idea

- Let r_i be the challenges of the verifier.
- Let m be the number of S-boxes, (i.e 10?/party).
  - Choose m_1, m_2 so that: $m = m_1 \cdot m_2$.
  - Divide the vector $(s_1^{(i)}, \ldots, s_m^{(i)})$ into m_2 chunks of size m_1.

$$s_k^{'(i)} = (r_1 s_{1+(k-1)m_\mathbf{1}}, \ldots, r_j s_{m_\mathbf{1}+(k-1)m_\mathbf{1}}) \;\; \forall k \in [m_2]$$

  - Similar for t.

$$t_k^{'(i)} = (t_{1+(k-1)m_\mathbf{1}}, \ldots, t_{m_\mathbf{1}+(k-1)m_\mathbf{1}}) \;\; \forall k \in [m_2]$$

  - The point:

$$\left\langle s_k^{'(i)}, t_k^{'(i)} \right\rangle = \sum_{j \in [m_\mathbf{1}]} r_j, \;\; \forall k \in [m_2]$$

- Build polynomials, P_j, S_j and T_j so that:

$$(S_1(k), \dots, S_{m_2}(k)) = s_k'^{(i)};\ (T_1(k), \dots, T_{m_2}(k)) = t_k'^{(i)};$$

$S_k(m_2)$, $T_k(m_2)$ to be chosen at random.

$$P = \sum_{j \in [m_1]} S_j T_j, \ \ \forall k \in [m_2]$$

And $S(l)$, $T(l)$ are chosen at random. And $P(l) = S(l) \cdot T(l)$.

- Lift P, S, T to a field $\mathbb{F}_{2^\lambda}$ with λ to be chosen large enough.
  - Choosing λ is not free as $dim(\mathbb{F}_{2^\lambda}/\mathbb{F}_2) = \lambda$. So \$$\lambda$\$-bits of information are needed per element in the field. $(3\lambda \cdot (m+1)$ for the polies for example).
  - Choose if $R \in \mathbb{F}_{2^\lambda} - [m_2]$ at random. Check $P(R) = S(R) \cdot T(R)$.

Sacrificing: Banquet. Some remarks.

- `Observation`: We can't choose $R \in [m_2]$ because at those points the equality is guaranteed to hold by construction.
- `Observation`: Interpolation is linear, hence the parties can compute a share of $S_j(R)$ without comunication.

Banquet cost.

- Complexity: $3\kappa + \tau \cdot (4\kappa + \kappa\lceil \log_2(N) \rceil + \mathcal{M}(C))$
  Around 20KB for $L1$.

Banquet soundness:

There are 2 ways an attacker might cheat: The cheater (prover) can cheat by guessing any ot the challenges in a given paralell repetition.

- He can make a guess on the first challenge (*varepsilon*). If he cheats on M_1 rounds we get:

$$P_1 = P(\text{Prover cheats this way}) = \sum_{k=M_1}^{M} PMF(k, M, 2^{-8\lambda})$$

$$PMF(k, M, p) = \binom{M}{k} p^k (1-p)^{M-k}$$

- He can make a guess on the second challenge (polynomial verification). If he cheats on M_2 rounds we get:

$$P_2 = P(\text{Prover cheats this way}) = \sum_{k=M_2}^{M-M1} PMF(k, M-M1, \frac{2m_2}{2^{8\lambda}-m_2})$$

Banquet soundness:

- He can make a guess on the first challenge (*varepsilon*). If he cheats on M_1 rounds we get:

$$P_1 = P(\text{Prover cheats this way}) = \sum_{k=M_1}^{M} PMF(k, M, 2^{-8\lambda})$$

$$PMF(k, M, p) = \binom{M}{k} p^k (1-p)^{M-k}$$

- He can make a guess on the second challenge (polynomial verification). If he cheats on M_2 rounds we get:

$$P_2 = P(\text{Prover cheats this way}) = \sum_{k=M_2}^{M-M1} PMF(k, M-M1, \frac{2m_2}{2^{8\lambda}-m_2})$$

- He can make a guess on the third challenge (unopeneded party is corrupted). If he cheats on M_3 rounds we get:

$$P_3 = P(\text{Prover cheats this way}) = N^{-M_3}$$

Banquet soundness:

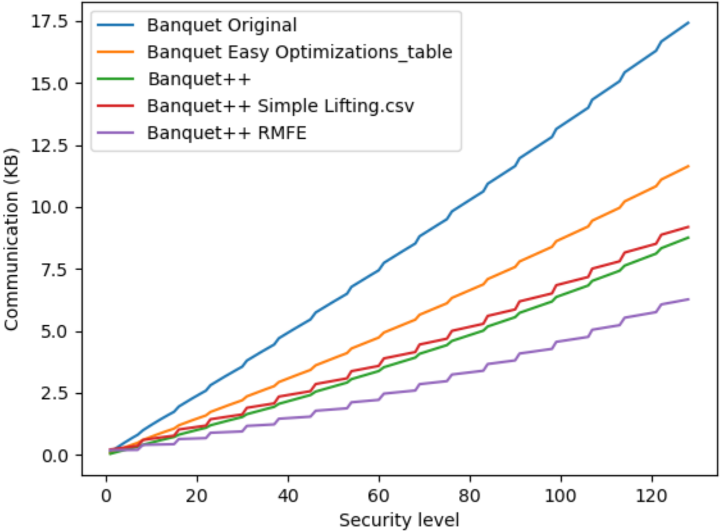
Then we must choose our parameters so that:

$$\frac{1}{P_1} + \frac{1}{P_2} + \frac{1}{P_3} > 2^{\kappa}$$

Subject to:

$$M = M_1 + M_2 + M_3$$

Some graphical comparisons.



Banquet proof

Theorem

Security Suposssing that Commit and H are random oracles: Banquet is EUF-CMA secure.

The idea is to prove first:

Theorem

Security Banquet is EUF-KO secure.

BN++: Better polynomial checking.

- Complexity complexity:

Protocol	Complexity
BN	$5C\log_2(\lvert\mathbb{F}\rvert)$
BN++ + basic optimization	$(2C+1)\log_2(\lvert\mathbb{F}\rvert)$
BN++ + simple lifting	$C\log_2(\lvert\mathbb{F}\rvert)+(C+1)\log_2(\lvert\mathbb{K}\rvert)$
BN++ + RMFE	$\left(2\left\lceil\frac{C}{k}\right\rceil+1\right)(\log_2(\lvert\mathbb{K}\rvert))$

BN++ Suggested optimizations.

One very simple optimizations based on the following obsevations: If x is a value in the circuit which is known, and $(x_i)_{i=0}^{N}$ is a secret sharing of x. Then there is no need for communication, since all-but-one shares can be computed by the verifier, and the other can be computed in the following way, if $\bar{i_e}$ is the unopened party:

$$x_{\bar{i_e}} = x - \sum_{i \in [N], i \neq \bar{i_e}} x_i$$

Based on this there are 2 savings:

- No need to send to V the output gate of $\bar{i_e}$.
  - Saves 1 element global.
- No need to send $v_{\bar{i_e}}$ (the verification element).
  - Saves 1 element/multiplication gate.

BN++ Suggested optimizations.

- No need to transmit β when sacrificing! Idea: Sacrificied triple can be correlated. We set $b_i = -y_i$, which can be done locally.

  Then we can do the sacrificing as usual, but $\beta = 0$.

  Then the sacrificied triple is $(a, -y, c)$ and a sharing $(a_i, -y_i, c_i)$ then choose ε at random and do:

  1. Compute $\alpha_i = \varepsilon x_i - a_i$
  2. Open α ($\beta = 0$), i.e
  $\alpha = \sum \alpha_i$
  3. Compute:
  $v_i = \varepsilon z_i - c_i - \alpha \cdot b_i$
  Then $v = \sum v_i = \varepsilon \Delta_z - \Delta_c$
  $\Delta_z = \sum z_i - \sum x_i \sum y_i$, $\Delta_c = \sum c_i - \sum a_i \sum b_i$
  4. Check that $v = 0$.

BN++ Suggested optimizations.

Batched and compressed checking

Kales and Zaverucha take credit for these, but this seems to be the combination of two methods suggested in Baum-Nof. Namely: Batched checking and output compression. Also same idea as in Banquet, only in Banquet polynomial checking is used for better soundness.

`Idea:`

Let $(x_i^j, y_i^j, z_i^j)_{i=0}^N$ be the multiplication triple of the j -th gate, so $\sum_i z_i^j = \sum_i x_i^j \sum y_i^j, \ \forall 1 \leq j \leq |C|$.

Then we want to verify that:

$$\left(x^1, \ldots, x^{|C|}\right) \cdot \left(y^1, \ldots, y^{|C|}\right) = \sum_{i,j} z_i^j = \sum_j z^j$$

BN++ Suggested optimizations.

Then we can check

$\sum z_i = (\sum x_i)(\sum y_i)$, without sharing x or y.
Then the sacrificied triple is $(a, -y, c)$ and a sharing $(a_i, -y_i, c_i)$ then
Choose ε^j challenges at random and compute:

1. Compute $\alpha_i^j = \varepsilon^j x_i^j - a_i^j$
2. Open α^j $(\beta^j = 0)$, i.e
$\alpha^j = \sum \alpha_i^j$
3. Compute:
$v_i = \sum_j (\varepsilon^j z_i^j - \alpha^j \cdot b_i^j) - c_i$
Then $v = \sum v_i = \varepsilon \Delta_z - \Delta_c$
$\Delta_z = \sum z_i - \sum x_i \sum y_i$, $\Delta_c = \sum c_i - \sum a_i \sum b_i$
4. Check that $v = 0$.

RMFE: Another idea from Banquet++

To lift an element from $\mathbb{F}_2$ to $\mathbb{F}_{2^8}$ requires 8 bits of communication.
We say that the rate of this embedding is 8. But we can do better.
Example: It is possible to lift 3 elements from $\mathbb{F}_2$ to $\mathbb{F}_{2^5}$ with a rate of 1.6 bits per element (instead of the usual 5).

BN++ The global picture

$\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$:

Phase 1: Committing to the seeds and views of the parties.

Sample a random salt: $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$.
for each parallel repetition e **do**
    Sample a root seed: $\mathsf{seed}_e \xleftarrow{\$} \{0,1\}^{\kappa}$.
    Derive $\mathsf{seed}_e^{(1)}, \dots, \mathsf{seed}_e^{(N)}$ as leaves of a binary tree from seed_e.
    for each party i **do**
        Commit to seed: $\mathsf{com}_e^{(i)} \leftarrow \mathsf{Commit}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$.
        Expand random tape: $\mathsf{tape}_e^{(i)} \leftarrow \mathsf{ExpandTape}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$
        Sample witness share: $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
    Compute witness offset: $\Delta\mathsf{sk}_e \leftarrow \mathsf{sk} - \sum_i \mathsf{sk}_e^{(i)}$.
    Adjust first share: $\mathsf{sk}_e^{(1)} \leftarrow \mathsf{sk}_e^{(1)} + \Delta\mathsf{sk}_e$.
    for each gate g in $\mathcal{C}$ with index ℓ **do**
        if g is an addition gate with inputs (x, y) **then**
            Party i locally computes the output share $z^{(i)} = x^{(i)} + y^{(i)}$.
        if g is a multiplication gate with inputs $(x_{e,\ell}, y_{e,\ell})$ **then**
            Compute output shares $z_{e,\ell}^{(i)} = \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
            Compute offset $\Delta z_{e,\ell} = x_{e,\ell} \cdot y_{e,\ell} - \sum_{i=1}^{N} z_{e,\ell}^{(i)}$.
            Adjust first share $z_{e,\ell}^{(1)} \leftarrow z_{e,\ell}^{(1)} + \Delta z_{e,\ell}$.
            For each party i, set $a_{e,\ell}^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
            Compute $a_{e,\ell} = \sum_{i=1}^{N} a_{e,\ell}^{(i)}$ and set $b_{e,\ell} = y_{e,\ell}$.
    Compute $c_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
    Compute offset $\Delta c_e = \left(\sum_{\ell=1}^{|C|} a_{e,\ell} \cdot b_{e,\ell}\right) - c_e$.
    Adjust first share: $c_e^{(1)} \leftarrow c_e^{(1)} + \Delta c_e$

23: Let $\mathsf{ct}_e^{(i)}$ be the output shares of online simulation.

24: Set $\sigma_1 \leftarrow (\mathsf{salt}, ((\mathsf{com}_e^{(i)}, \mathsf{ct}_e^{(i)})_{i\in[N]}, \Delta\mathsf{sk}_e, \Delta c_e, (\Delta z_{e,\ell})_{\ell\in[C]})_{e\in[\tau]}$.

Phase 2: Challenging the checking protocol.

1: Compute challenge hash: $h_1 \leftarrow H_1(\mathsf{salt}, \mathsf{msg}, \sigma_1)$.

2: Expand hash: $((\epsilon_{e,\ell})_{\ell\in[C]})_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_1)$ where $\epsilon_{e,\ell} \in \mathbb{F}$.

Phase 3: Commit to simulation of the checking protocol.

1: **for** each repetition e **do**

2: For the set of multiplication gates in $\mathcal{C}$, simulate the triple checking protocol as defined in §2.6 for all parties with challenge $(\epsilon_{e,\ell})_{\ell\in[C]}$. The inputs are $(x_{e,\ell}^{(i)}, y_{e,\ell}^{(i)}, z_{e,\ell}^{(i)}, a_{e,\ell}^{(i)}, b_{e,\ell}^{(i)}, c_e^{(i)})$, and let $\alpha_{e,\ell}^{(i)}$ and $v_e^{(i)}$ be the broadcast values.

3: Set $\sigma_2 \leftarrow (\mathsf{salt}, (((\alpha_{e,\ell}^{(i)})_{\ell\in[C]}, v_e^{(i)})_{i\in[N]})_{e\in[\tau]}$.

Phase 4: Challenging the views of the MPC protocol.

1: Compute challenge hash: $h_2 \leftarrow H_2(h_1, \sigma_2)$.

2: Expand hash: $(\bar{i}_e)_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_2)$ where $\bar{i}_e \in [N]$.

Phase 5: Opening the views of the MPC and checking protocols.

1: **for** each repetition e **do**

2: $\mathsf{seeds}_e \leftarrow \{\log_2(N)$ nodes to compute $\mathsf{seed}_e^{(i)}$ for $i \in [N] \setminus \{\bar{i}_e\}\}$.

3: Output $\sigma \leftarrow (\mathsf{salt}, h_1, h_2, (\mathsf{seeds}_e, \mathsf{com}_e^{(\bar{i}_e)}, \Delta\mathsf{sk}_e, \Delta c_e, (\Delta z_{e,\ell}, \alpha_{e,\ell}^{(\bar{i}_e)})_{\ell\in[C]})_{e\in[\tau]})$.

BN++ The global picture

$\mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma)$:

Parse σ as $(\mathsf{salt}, h_1, h_2, (\mathsf{seeds}_e, \mathsf{com}_e^{(\bar{i}_e)}, \Delta\mathsf{sk}_e, \Delta c_e, (\Delta z_{e,\ell}, \alpha_{e,\ell}^{(\bar{i}_e)})_{\ell\in[C]})_{e\in[\tau]})$.
Expand hashes: $(\epsilon_{e,\ell})_{e\in[\tau],\ell\in[C]} \leftarrow \mathsf{Expand}(h_1)$, and $(\bar{i}_e)_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_2)$.
for each repetition e **do**
    Use seeds_e to recompute $\mathsf{seed}_e^{(i)}$ for $i \in [N] \setminus \bar{i}_e$.
    for each party $i \in [N] \setminus \bar{i}_e$ **do**
        Recompute $\mathsf{com}_e^{(i)} \leftarrow \mathsf{Commit}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$,
        $\mathsf{tape}_e^{(i)} \leftarrow \mathsf{ExpandTape}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$, and
        $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
        if $i = 1$ **then** adjust first share: $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{sk}_e^{(i)} + \Delta\mathsf{sk}_e$.
        for each gate g in $\mathcal{C}$ with index ℓ **do**
            if g is an addition gate with inputs $(x^{(i)}, y^{(i)})$ **then**
                Compute the output share $z^{(i)} = x^{(i)} + y^{(i)}$
            if g is a mult. gate, with inputs $(x_{e,\ell}^{(i)}, y_{e,\ell}^{(i)})$ **then**
                Compute output share $z_{e,\ell}^{(i)} = \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
                if $i = 1$ **then**
                    Adjust first share $z_{e,\ell}^{(i)} \leftarrow z_{e,\ell}^{(i)} + \Delta z_{e,\ell}$.
                Set $a_{e,\ell}^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$, and $b_{e,\ell}^{(i)} = y_{e,\ell}^{(i)}$
        Set $c_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$
        if $i = 1$ **then** adjust first share $c_e^{(i)} \leftarrow c_e^{(i)} + \Delta c_e$.

Let $\mathsf{ct}_e^{(i)}$ be party i's share of the circuit output.

Compute $\mathsf{ct}_e^{(\bar{i}_e)} = \mathsf{ct} - \sum_{i \neq \bar{i}_e} \mathsf{ct}_e^{(i)}$

Set $\sigma_1 \leftarrow (\mathsf{salt}, ((\mathsf{com}_e^{(i)}, \mathsf{ct}_e^{(i)})_{i \in [N]}, \Delta\mathsf{sk}_e, \Delta c_e, (\Delta z_{e,\ell})_{\ell \in [C]})_{e \in [\tau]})$.

Set $h_1' = H_1(\mathsf{salt}, \mathsf{msg}, \sigma_1)$

for each repetition e **do**

for each party $i \in [N] \setminus \bar{i}_e$ **do**

For the set of multiplication gates in $\mathcal{C}$, simulate the triple verification procedure as defined in §2.6 for party i with challenge $(\epsilon_{e,\ell})_{\ell \in [C]}$. The inputs are $(x_{e,\ell}^{(i)}, y_{e,\ell}^{(i)}, z_{e,\ell}^{(i)}, a_{e,\ell}^{(i)}, b_{e,\ell}^{(i)}, c_e^{(i)})$, and let $\alpha_{e,\ell}^{(i)}$ and $v_e^{(i)}$ be the broadcast values.

Compute $v_e^{(\bar{i}_e)} = 0 - \sum_{i \neq \bar{i}_e} v_e^{(i)}$

Set $\sigma_2 \leftarrow (\mathsf{salt}, (((\alpha_{e,\ell}^{(i)})_{\ell \in [C]}, v_e^{(i)})_{i \in [N]})_{e \in [\tau]}$.

Set $h_2' = H_2(h_1', \sigma_2)$.

Output accept iff $h_1' \stackrel{?}{=} h_1$ and $h_2' \stackrel{?}{=} h_2$.

Some delicate stuff.

- Since only one plaintext/ciphertext pair is revealed, less rounds are necessary to guarantee 128 bits of security.
- Rain: Bigger S-box allows to work in larger fields.

Ligero

Limbo

Wolverine

1.
2.

Mac'n'cheese

Unexplored options

- NIMPC: Probably really bad.
- Could the last party proof it's been honest?
  - Already some work on this.
  - Paper of Nof proving that: If the sharing has some redundancy, then the one party has low chance of cheating. Problem: Currently it requires leaving 2 parties unopened. What we need: Some sort of magical homomorphic (1-time) MAC.
- AES-specific sacrificing.
- Multlivariate Schwartz-Zippel.
- Multiple verifiers!.
  - Used in MPC for better performance, could be ported to MPCiTH for the non-interactive (our) case.
- Sampling on the fly.

Alternatives.

- Binary-SIS
- Random codes.
- Trapdoorless multivariate?

u