Multi-Key Shuffle: Network Application for Verifier

Author: Aaron Kimmig <kimmiga@informatik.uni-freiburg.de> <academia@aaronkimmig.de>
Last Edited: 2023-12-11

Implementing verifier component in network application for efficient Multi-Key Shuffle Argument

For a fast start, go to section "Configuration" > "Set Notebook Context and Kernel".

See
[Master's Thesis] Efficient Zero-Knowledge Multi-Key Verifiable Shuffles: Implementation, Security and Applications
[BünzDraft23] Bünz, B., Raykova, M., & Sarathy, J. (2023, Draft). Efficient Multi-Key Verifiable Shuffles from Short Arguments for Randomized Algorithms.
for Shuffle Argument and
[Bünz18] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., & Maxwell, G. (2018). Bulletproofs: Short proofs for confidential transactions and more. In 2018 IEEE Symposium on Security and Privacy (SP) (pp. 315-334). IEEE.
for Inner Product argument.

Configuration options:

Crypto & Protocol
- `interactive:` Whether to use interactive protocol or Fiat-Shamir transform
- `useInnerProductProtocol:` Whether to use the communication-cost-efficient inner product protocol
- `rngMethod:` "OpenSSL", "ExtendedCA", ... (All of Mathematica's options)
- `fixedNUMSRandomSeed:` Fixed random seed which will be used for calculating the generators for Pedersen Commitments
- `securityPolicy:` required minimum bits for Elliptic Curve and ElGamal primes
- `setupsPermitted:` selection of setups that are accepted for verification (additionally, entropy policy has to be fulfilled)

By Application of Prover's Setup
- `useElGamal:` Choose between El Gamal and Elliptic Curve Cryptography (only EC is implemented; configuration option left for demonstration of El Gamal setup)
- `p, q, (implicit: cofactor), g:` Trapdoor Function / Schnorr Group

– `p, n_EC, h_EC, (implicit: q), a_EC, b_EC, g:` Trapdoor Function / Elliptic Curve (Weierstrass representation)

Network, Communication & Environment

– `myId`: Choose a name that identifies the verifier. Could be replaced by a public key if a public key infrastructure was implemented
– `proverId:` ID of prover
– `room:` "address" of the proof and where messages are routed over
– `roomPassword:` password needed to access the room
– `inspectionEnabled`: Publish variables to Inspector for educational purposes
– `routerConfig`: Connection to message router and inspector
– `useLocalRouter`: Whether to use a self-hosted local router instance or a public router server on the internet
– `fiatShamirSingleKernel:` Set to True when using a mathematica test license (which allows only one kernel to be run simultaneously) and not running prover and verifier on different machines
– `verifierKernelName:` Name of kernel this notebook shall be run on if fiatShamirSingleKernel = False
– `verbosity settings:` Set verbosity levels for different components

Prover's Notebook: "MultiKeyShuffle-Prover.nb"

Start the router and inspector before running this notebook when using a local router:
```
$ python3 RoomServer.py
```
To get live inspection, visit http://localhost:11080 when running your local server or https://aaronkimmig.de/crypto-insight if you want to use the standard external server.

Reset – Clean up Notebook, Clear states

If fiatShamirSingleKernel is set in section "Configuration" run CLEAR ALL after the prover's notebook has been run and before this notebook is being executed!

CREATE NEW KERNEL IF NEEDED:

Menu **>** Evaluation > Kernel Configuration Options > Add > Name: Verifier > OK

The following helper actions are useful for cleanup and debugging. They are not required to correctly execute the protocols.

CLEAR ALL

In[ ]:=
```
FrontEndTokenExecute["DeleteGeneratedCells"];
Print["Deleted all generated output cells"];
Print["Quit Kernel"];
Quit[];
```

```
Deleted all generated output cells

Quit Kernel
```

LEAVE ROOM (Can only be executed after initialization)

In[ ]:=
```
SendVariables["Router", {"__leave_room__" → True}];
WaitForVariable["__leave_room__", "Router"]
```

RECONNECT (Can only be executed after initialization)

In[ ]:=
```
(*Reconnect with new socket*)
router = ConnectToRouter[routerConfig, True];
```

CLEAR SAVED STATES (Can only be executed after initialization)

```
StateMachineClear[];
```

Configuration

Set Notebook Context and Kernel

- Using a payed license?
- YES: On your first run, make sure the kernel for the prover has been created: Menu **>** Evaluation > Kernel Configuration Options > Add > Name: Prover > OK
- NO: set option fiatShamirSingleKernel = True and read the instructions in the comment above it.

- To run the whole notebook, place the cursor into this cell, press “Shift-Enter” and click “Yes” to evaluate the whole notebook. Select “No” to only evaluate this cell. Go on to the next cells to run them one by one with “Shift-Enter”.

- To clean up the notebook and start with a new run, execute the first cell (“Shift-Enter”) in section “Reset”. Re-evaluating cells is possible because a state machine allows protocol rewinding.

Re-evaluate after changes

In[325]:=

```
(*Whether to run notbook on default kernel.
Useful with Mathematica Test License where only a single kernel can be run at a time.
Remark: Running the interactive protocol is not possible on a single kernel!
Also apply this setting to the prover and run it before the verifier.
Quit Kernel after prover has been run and before verifier is started!
interactive = False has to be set in the configuration.*)
fiatShamirSingleKernel = False;(*Default: False*)
(*Name of kernel this notebook should be run on.
Does not take effect if fiatShamirSingleKernel = True (the notebook will then be run on Kernel
verifierKernelName = "Verifier";(*Default: "Verifier"*)

(*=== don't change between here >>> ===*)
(*Context*)
Print["Setting Cell Context to Notebook only"]
SetOptions[EvaluationNotebook[],CellContext→Notebook];
(*Kernel*)
kernelName = If[fiatShamirSingleKernel,"Local",verifierKernelName];
(*Make sure that a kernel with this name has been created via Menu→Evaluation→Kernel Configur
SetOptions[EvaluationNotebook[],Evaluator→kernelName];
Print["Evaluating on Kernel named \""<>kernelName<>"\""]
(*=== <<< and here ===*)
```

```
Setting Cell Context to Notebook only

Evaluating on Kernel named "Verifier"
```

Crypto & Protocol

In[8]:=

```
(*Whether to use the interactive protocol or apply Fiat-Shamir transform: True/False*)
interactive = True;
(*Whether to use the inner product protocol for logarithmic communication cost; set to True if
useInnerProductProtocol = True;(*Default: True*)
(*How random numbers should be generated, choose from
"ExtendedCA" (built-in pseudo random number generator) or
"OpenSSL", (cryptographic random number generator from OpenSSL)
(*to be implemented: "DevRandom" (use system entropy from /dev/random)*)
This setting is automatically set to "ExtendedCA" if a fixed random seed is used for the verif
rngMethod = "OpenSSL";(*Default: "OpenSSL"*)
(*Fixed random seed which will be used for calculating the generators for Pedersen Commitments
fixedNUMSRandomSeed = 314159000001;(*Default: 314159000001*)
(*Specify minimum bits for group elements and scalars*)
securityPolicy = <|
    "EC"→<|"group"→256,"scalar"→256|>,
    "ElGamal"→<|"group"→2048,"scalar"→256|>
|>;
(*specify selection of setups that are accepted for verification; empty to allow all
Additionally, entropy policy has to be fulfilled*)
setupsPermitted = <|
    (*only support secp256k1*)
    "EC"→<|
        "SECP256K1"→<|
            "curve_representation"→"Weierstrass",
            "a_EC"→0,
            "b_EC"→7,
            "p"→2^256-2^32-2^9-2^8-2^7-2^6-2^4-1,
            "h_EC"→1
        |>
    |>,
    (*support all setups that conform to entropy policy*)
    "ElGamal"→<||>
|>;
```

Network, Communication & Environment

In[14]:=
```
(*Unique ID*)
myId = "Alice-Shuffle-Ver";
(*ID of prover*)
proverId = "Bob-Shuffle-Pro";
(*Room the communication is routed over*)
room = "BobsBPShuffle2";
roomPassword = "mediumBPShuffle";
(*Enable/Disable publishing (secret) variables to Inspector for educational purposes*)
inspectionEnabled = True;(*Default: True*)
(*Connection to router*)
useLocalRouter = False;
routerConfig = <|
    "Host"→If[useLocalRouter,"127.0.0.1",TextString[HostLookup["aaronkimmig.de"][[1]][[1]]],
    "Port"→25080,
    "Password"→"Drehkreuz64",
    "ConnectionPassword"→"Call me Alice"
|>;
(*Verbosity Levels: 0, 1, 2*)
stateMachineVerbosity = 0;
fiatShamirVerbosity = 0;
```

Initialization and Setup

Symbolize all Variables with Special Appearance

In[23]:=
```
(*Load notation package for pretty variables*)
Needs["Notation`"]
SymbolQ = MatchQ[#, t_Symbol /; AtomQ[t]] &;
If[SymbolQ[guard_symbolize],
Print["Special symbols already set up"],
(*Else: Symbolize variables with special appearance*)
Symbolize[ guard_symbolize ];
(*Elliptic curve*)
Symbolize[ a_EC ];
Symbolize[ b_EC ];
Symbolize[ h_EC ];
Symbolize[ n_EC ];
(*generators*)
Symbolize[ g⃗ ];
(*public input and output values to shuffle argument*)
Symbolize[ h⃗ ];
Symbolize[ b⃗ ];
```

```
Symbolize[ c⃗ ];
Symbolize[ d⃗ ];
Symbolize[ h⃗' ];
Symbolize[ b⃗' ];
Symbolize[ c⃗' ];
Symbolize[ d⃗' ];
Symbolize[ h⃗* ];
Symbolize[ b⃗* ];
Symbolize[ c⃗* ];
Symbolize[ d⃗* ];
(*helper scalars and vectors*)
Symbolize[ v⃗_0^n ];
Symbolize[ v⃗_1^n ];
Symbolize[ v⃗_2^n ];
Symbolize[ y⃗^n ];
Symbolize[ y⃗_z^(n-1) ];
Symbolize[ y^-1 ];
Symbolize[ y^-n⃗ ];
(*shuffle protocol*)
Symbolize[ k⃗ ];
Symbolize[ k̂ ];
Symbolize[ k̄ ];
Symbolize[ A_R,h ];
Symbolize[ A_R,b ];
Symbolize[ A_R,c ];
Symbolize[ A_R,d ];
Symbolize[ A_L ];
Symbolize[ S_L ];
Symbolize[ S_R,h ];
Symbolize[ S_R,b ];
```

```
Symbolize[ S_{R,c} ];
Symbolize[ S_{R,d} ];
Symbolize[ \vec{l} ];
Symbolize[ \vec{r} ];
Symbolize[ τ_x ];
Symbolize[ T_3 ];
Symbolize[ T_5 ];
Symbolize[ δ_{yz} ];
Symbolize[ \hat{t} ];
Symbolize[ μ_h ];
Symbolize[ μ_b ];
Symbolize[ μ_c ];
Symbolize[ μ_d ];
Symbolize[ P_h^+ ];
Symbolize[ P_b^+ ];
Symbolize[ P_c^+ ];
Symbolize[ P_d^+ ];
(*PR protocol*)
Symbolize[ A_W ];
Symbolize[ S_W ];
Symbolize[ δ_{yzw} ];
Symbolize[ t_{0,W} ];
Symbolize[ t_{1,W} ];
Symbolize[ t_{2,W} ];
Symbolize[ \vec{l}_W ];
Symbolize[ \vec{r}_W ];
Symbolize[ T_{1,W} ];
Symbolize[ T_{2,W} ];
Symbolize[ \hat{t}_W ];
Symbolize[ μ_W ];
Symbolize[ P_W^+ ];
(*inspection*)
```

```
Symbolize[ π^-1 ];
(*Symbolize[ σ_V ];*)
Symbolize[ σ_W ];
Symbolize[ a_L→ ];
Symbolize[ a_R→ ];
Symbolize[ s_L→ ];
Symbolize[ s_R→ ];
Symbolize[ ρ_h ];
Symbolize[ ρ_b ];
Symbolize[ ρ_c ];
Symbolize[ l_0→ ];
Symbolize[ l_2→ ];
Symbolize[ r_1→ ];
Symbolize[ r_3→ ];
Symbolize[ t_3 ];
Symbolize[ t_5 ];
Symbolize[ τ_3 ];
Symbolize[ τ_5 ];
(*inspection: PR*)
Symbolize[ a_L,W→ ];
Symbolize[ a_R,W→ ];
Symbolize[ s_L,W→ ];
Symbolize[ s_R,W→ ];
Symbolize[ α_W ];
Symbolize[ β_W ];
Symbolize[ γ_W ];
Symbolize[ l_0,W→ ];
Symbolize[ l_1,W→ ];
Symbolize[ r_0,W→ ];
Symbolize[ r_1,W→ ];
Symbolize[ τ_1,W ];
```

```
Symbolize[ τ_{2,W} ];
Symbolize[ τ_{x,W} ];
(*inner product protocol*)
Symbolize[ n_2 ];
Symbolize[ n_{pow2} ];
Symbolize[ g' ];
Symbolize[ u_{IP} ];
Symbolize[ n_0 ];
Symbolize[ k_2 ];
Symbolize[ k_0 ];
Symbolize[ P_h^* ];
Symbolize[ P_b^* ];
Symbolize[ P_c^* ];
Symbolize[ P_d^* ];
Symbolize[ P_W^* ];(*PR*)
Symbolize[ P_h' ];
Symbolize[ P_b' ];
Symbolize[ P_c' ];
Symbolize[ P_d' ];
Symbolize[ P_W' ];(*PR*)
Symbolize[ P_{2,h} ];
Symbolize[ P_{2,b} ];
Symbolize[ P_{2,c} ];
Symbolize[ P_{2,d} ];
Symbolize[ P_{2,W} ];(*PR*)
Symbolize[ P_{next,h} ];
Symbolize[ P_{next,b} ];
Symbolize[ P_{next,c} ];
Symbolize[ P_{next,d} ];
Symbolize[ P_{next,W} ];(*PR*)
Symbolize[ \vec{G}_2 ];
Symbolize[ \vec{G} ];
```

```
Symbolize[ $\overrightarrow{G_{lo}}$ ];
Symbolize[ $\overrightarrow{G_{hi}}$ ];
Symbolize[ $\overrightarrow{G_{next}}$ ];
Symbolize[ $\overrightarrow{H_2}$ ];
Symbolize[ $\vec{H}$ ];
Symbolize[ $\overrightarrow{H_{lo}}$ ];
Symbolize[ $\overrightarrow{H_{hi}}$ ];
Symbolize[ $\overrightarrow{H_{next}}$ ];
Symbolize[ $\overrightarrow{B_2}$ ];
Symbolize[ $\vec{B}$ ];
Symbolize[ $\overrightarrow{B_{lo}}$ ];
Symbolize[ $\overrightarrow{B_{hi}}$ ];
Symbolize[ $\overrightarrow{B_{next}}$ ];
Symbolize[ $\overrightarrow{C_2}$ ];
Symbolize[ $\vec{C}$ ];
Symbolize[ $\overrightarrow{C_{lo}}$ ];
Symbolize[ $\overrightarrow{C_{hi}}$ ];
Symbolize[ $\overrightarrow{C_{next}}$ ];
Symbolize[ $\overrightarrow{D_2}$ ];
Symbolize[ $\vec{D}$ ];
Symbolize[ $\overrightarrow{D_{lo}}$ ];
Symbolize[ $\overrightarrow{D_{hi}}$ ];
Symbolize[ $\overrightarrow{D_{next}}$ ];
Symbolize[ $L_h$ ];
Symbolize[ $L_b$ ];
Symbolize[ $L_c$ ];
Symbolize[ $L_d$ ];
Symbolize[ $L_W$ ];(*PR*)
Symbolize[ $L_{2,h}$ ];
```

```
Symbolize[ L_{2,b} ];
Symbolize[ L_{2,c} ];
Symbolize[ L_{2,d} ];
Symbolize[ L_{2,W} ];(*PR*)
Symbolize[ R_h ];
Symbolize[ R_b ];
Symbolize[ R_c ];
Symbolize[ R_d ];
Symbolize[ R_W ];(*PR*)
Symbolize[ R_{2,h} ];
Symbolize[ R_{2,b} ];
Symbolize[ R_{2,c} ];
Symbolize[ R_{2,d} ];
Symbolize[ R_{2,W} ];(*PR*)
Symbolize[ u_2 ];
Symbolize[ u^{-1} ];
Symbolize[ a_0 ];
Symbolize[ b_0 ];
Symbolize[ c_0 ];
Symbolize[ a_{0,W} ];(*PR*)
Symbolize[ b_{0,W} ];(*PR*)
Symbolize[ c_{0,W} ];(*PR*)
Symbolize[ G_0 ];
Symbolize[ H_0 ];
Symbolize[ B_0 ];
Symbolize[ C_0 ];
Symbolize[ D_0 ];
Symbolize[ P_{0,h} ];
Symbolize[ P_{0,b} ];
Symbolize[ P_{0,c} ];
Symbolize[ P_{0,d} ];
Symbolize[ P_{0,W} ];(*PR*)
```

```
(*inspection*)
Symbolize[ $\overrightarrow{a_{IP}}$ ];
Symbolize[ $\overrightarrow{a_{lo}}$ ];
Symbolize[ $\overrightarrow{a_{hi}}$ ];
Symbolize[ $\overrightarrow{b_{IP}}$ ];
Symbolize[ $\overrightarrow{b_{lo}}$ ];
Symbolize[ $\overrightarrow{b_{hi}}$ ];
Symbolize[ $\overrightarrow{a_{W}}$ ];(*PR*)
Symbolize[ $\overrightarrow{a_{lo,W}}$ ];(*PR*)
Symbolize[ $\overrightarrow{a_{hi,W}}$ ];(*PR*)
Symbolize[ $\overrightarrow{b_{W}}$ ];(*PR*)
Symbolize[ $\overrightarrow{b_{lo,W}}$ ];(*PR*)
Symbolize[ $\overrightarrow{b_{hi,W}}$ ];(*PR*)
Print["Finished setting up special symbols"]
];
```

```
Finished setting up special symbols
```

Apply Configuration, Load Libraries, Set up Network and Communication

Re-evaluate after changes to configuration

In[26]:=

```
(*Check Configuration*)
If[fiatShamirSingleKernel,
    If[interactive,
        Print["fiatShamirSingleKernel only works with Fiat-Shamir activated, but it is not. Fo
        interactive = False;
    ];
];
(*Load ElGamal library*)
Print["Looking for SchnorrPrimes library: "<>TextString[FindFile["SchnorrPrimes`"]]];
Needs["SchnorrPrimes`"];
(*Load Elliptic Curve library*)
Print["Looking for elliptic curve library with overloaded + and * operators (ECOverloaded): "
Needs["ECOverloaded`"];
(*Load Sigma Tools library*)
Print["Looking for Sigma Protocol Tools library (SigmaTools): "<>TextString[FindFile["SigmaTo
Needs["SigmaTools`"];
(*Fiat-Shamir*)
If[interactive,
    semanticsOfRandomChallenge = "Random Challenge";
    Print["Interactive mode. Sending challenge values to prover. Not using Fiat-Shamir heurist
(*Else*)
    semanticsOfRandomChallenge = "Fiat-Shamir transform\nof Random Challenge";
```

```
    Print["Looking for Fiat-Shamir library (FiatShamir): "<>TextString[FindFile["FiatShamir`"
    Needs["FiatShamir`"];
    Print["Non-interactive mode. Using Fiat-Shamir heuristic."]
];
(*Verification success/failure*)
VerificationSucceeded[] := Module[{},
    SendVerificationSucceeded[];
    Print["Verification succeeded."]
]
VerificationFailed[reason_String] := Module[{},
    SendVerificationFailed[reason];
    Print["Verification failed: "<>reason];
    FrontEndTokenExecute["EvaluatorAbort"]
]
(*Inspection*)
InspectionSucceeded[] := Module[{},
    SendInspectionSucceeded[];
    Print["Inspection succeeded."]
]
InspectionFailed[reason_String] := Module[{},
    SendInspectionFailed[reason];
    Print["Inspection failed: "<>reason];
]
(*General Error*)
Error[reason_:""] := Module[{},
    Print[If[reason=="","Error.","Error: "<>reason]];
    FrontEndTokenExecute["EvaluatorAbort"]
]
(*Random Number Generator*)
SetRNGMethod[rngMethod];
SetFixedRandomSeeds[False];
SeedRandom[Method→rngMethod];
SetStateMachineVerbosity[stateMachineVerbosity];
SetFiatShamirVerbosity[fiatShamirVerbosity];
(*Communication setup*)
SetMyRole[If[interactive,"InteractiveVerifier","Verifier"]];
SetMyId[myId];
SetRoom[room];
SetInspection[inspectionEnabled];
(*Network: set up connection to router / don't force reconnection / force authorization*)
router = ConnectToRouter[routerConfig, False, True];
StateMachineClear[];
If[interactive,
    (*Interactive*)
    (*Network: join room for interactive verification*)
    roomConfig = {
        "password"→roomPassword,
        "ownership_type"→"creator",
        "ids_allowed"→{proverId, myId},
        "role_restrictions"→{proverId→"Prover", myId→"InteractiveVerifier"},
        "creator"→proverId,
        "wait_if_non_existing"→{"timeout"→90},
        "external_keys"→{
            "_verification_result_",
            "_inspection_request_",
```

```
            "_inspection_result_"
        }
    };
    roomInfo = JoinRoomAndWaitUntilComplete[roomConfig];
    memberIds = roomInfo["members"];
    (*Verify room and members*)
    If[Length[memberIds]≠2,
        Print["Assertion failure: Expecting two members in room '"<>room<>"' but members are
    (*Else If*)
    If[!(memberIds[[1]]==myId && memberIds[[2]]==proverId) && !(memberIds[[1]]==proverId && memberIds
        Print["Assertion failure: Expecting to be member of room '"<>room<>"' but members are
    ]]
];
StateMachineSave["Init"];
```

```
Looking for SchnorrPrimes library:
  C:\Users\Aaron\AppData\Roaming\Mathematica\Applications\SchnorrPrimes.wl

Looking for elliptic curve library with overloaded + and * operators (ECOverloaded):
  C:\Users\Aaron\AppData\Roaming\Mathematica\Applications\ECOverloaded.wl

Looking for Sigma Protocol Tools library (SigmaTools):
  C:\Users\Aaron\AppData\Roaming\Mathematica\Applications\SigmaTools.wl

Interactive mode. Sending challenge values to prover. Not using Fiat-Shamir heuristic.

Network connection to router:
```

SocketObject[Foreign IPAddress: 5.189.143.94 Foreign Port: 25080 Protocol: TCP (Client) UUID: 752f5657-d717-4213-9093-25fb2d920a87]

```
Authorizing, waiting for confirmation ...

Waiting to receive __authorize__ ...

Handling Event: success: __authorize__ (Router): Authorization successful

Sending message:
InteractiveVerifier@Alice-Shuffle-Ver:Router:BobsBPShuffle2:json:{"__join_room__":{"
  password":"mediumBPShuffle","ownership_type":"creator","ids_allowed":["Bob-Shuffle-Pro","
  Alice-Shuffle-Ver"],"role_restrictions":{"Bob-Shuffle-Pro":"Prover","Alice-Shuffle-Ver":"
  InteractiveVerifier"},"creator":"Bob-Shuffle-Pro","wait_if_non_existing":{"timeout":90},"
  external_keys":["_verification_result_","_inspection_request_","_inspection_result_"]}}

Joining room 'BobsBPShuffle2'. Waiting for
  confirmation and for all members to have joined to room ...

Waiting to receive __join_room__, __room_complete__ ...

Handling Event: success: __join_room__ (Router): Joined room 'BobsBPShuffle2'

Handling Event: success: __room_complete__ (Router):
  Event Data: <|Length:1|>
  Keys: {__room_complete__, _Message, _Valid, _SenderRoleAtId,
  _SenderRole, _SenderId, _ReceiverRoleAtId, _ReceiverRole, _ReceiverId,
  _UniqueKey, _Room, _DataFormat, _Data, _UnderscoreVars1, _UnderscoreVars2}
```

Trapdoor Function: El Gamal or Elliptic Curve

Get and apply setup of crypto algorithm from prover.

In[52]:=

```
StateMachineRestore["Init"];
If[!interactive,
```

```
    (*Non-interactive*)
    SendVariables["Inspector",{"__get_public_room_info__"→True,"__room_password__"→roomPassw
    roomInfo = WaitForVariable["__get_public_room_info__"];
    proof = Join[roomInfo["proof"]];
    (*Check if proof is finished*)
    If[!KeyExistsQ[proof,If[useInnerProductProtocol,"b0","rvec"]<>":Prover@"<>EscapeColonsAnd
        Print[
            "Info: The prover has not yet added all required variables to the proof. "<>
            "Won't be able to do complete verification. Currently knowing: "<>
            ToString[Keys[proof]]
        ]
    ];
    proof["_cost"] = Map[#["entropy"]&,Select[roomInfo["variable_meta"], KeyExistsQ[#, "entro
    EnqueueAsUnreadVariables[proof];
    (*variables will now be accessible via WaitForProversVariable*)
];

setupParameters = WaitForProversVariable["setup"];
(*Assertions*)
If[setupParameters["room"] ≠ room, Error["Assertion failure: prover's room "<>setupParameters
If[setupParameters["prover"] ≠ proverId, Error["Assertion failure: prover's id "<>setupParame
If[interactive, If[setupParameters["verifier"] ≠ myId, Error["Assertion failure: verifier id "
If[setupParameters["interactive"] ≠ interactive, Error["Assertion failure: interactivity "<>To
n = setupParameters["n"];
If[!IntegerQ[n] || n ≤ 0, Error["Assertion failure: prover's number of elements to be shuffle
If[!interactive && setupParameters["nums_seed"] ≠ fixedNUMSRandomSeed, Error["Assertion failu
(*Trapdoor function*)
encryptionMethod = setupParameters["encryption_method"];
useElGamal = (encryptionMethod == "ElGamal");
CheckSetup[setupType_String] := Module[{permitted, permitted2, requiredValue},
    If[!MemberQ[setupsPermitted, setupType], Return[True]];
    If[Length[setupsPermitted[setupType]] == 0, Return[True]];
    permitted = False;
    Scan[Function[{setupName},
        permitted2 = True;
        Scan[Function[{configKey},
            If[!MemberQ[setupParameters, configKey], permitted2 = False; Return[]];
            requiredValue = setupsPermitted[setupName][configKey];
            If[ListQ[requiredValue], requiredValue = ecPnt[requiredValue[[1]], requiredValue[[2]]
            If[setupParameters[configKey] ≠ requiredValue, permitted2 = False];
        ], Keys[setupsPermitted[setupName]]];
        If[permitted2,
            Print["Prover's "<>setupType<>" setup matches "<>setupName<>" which is permitted"
            permitted = True;
        ];
    ], Keys[setupsPermitted]];
    If[permitted,
        True
    (*Else*),
        VerificationFailed[setupType<>" configuration does not match any of the ones permitte
        False
    ]
]
If[useElGamal,
```

```
    CheckSetup["ElGamal"];
    (*p, q, g*)
    p = setupParameters["crypt_p"];
    h = setupParameters["crypt_h"];
    g = setupParameters["crypt_g"];
    q = (p - 1) / h;
    pBits = Length[IntegerDigits[p,2]];
    qBits = Length[IntegerDigits[q,2]];
    modq = Mod[#,q]&;
    modinvq = ModularInverse[#,q]&;
    powermodq = PowerMod[#1,#2,q]&;
    modp = Mod[#,p]&;
    modinvp = ModularInverse[#,p]&;
    powermodp = PowerMod[#1,#2,p]&;
    Print[Grid[{
        {"p is prime",PrimeQ[p]},
        {"bits of p",pBits},
        {"q is prime",PrimeQ[q]},
        {"bits of q",qBits},
        {"bits of cofactor",Length[IntegerDigits[h,2]]},
        {"cofactor * q \[Congruent]? p - 1",h*q==p-1},
        {"g^q \[Congruent]?_p 1",PowerMod[g,q,p]==1}
    },Alignment→{{Left,Left}},Frame→All]];
    (*Selection of security and sanity checks*)
    If[!PrimeQ[p],VerificationFailed["Non-prime Schnorr group modulus p"]];
    If[!PrimeQ[q],VerificationFailed["Non-prime subgroup order q"]];
    If[!IntegerQ[h],VerificationFailed["Non-integer cofactor p/q"]];
    If[KeyExistsQ[setupParameters, "entropy_parameters"],
        entropyParameters = setupParameters["entropy_parameters"];
        If[pBits < entropyParameters["group"],Print["Warning: number of bits of p is smaller
        If[qBits < entropyParameters["scalar"],Print["Warning: number of bits of q is smaller
    (*Else*),
        entropyParameters = <|"group"→qBits,"scalar"→qBits|>;
    ];
    If[KeyExistsQ[setupParameters, "security_parameters"],
        securityParameters = setupParameters["security_parameters"];
        If[pBits ≠ securityParameters["group"],Print["Warning: number of bits of p does not m
        If[qBits ≠ securityParameters["scalar"],Print["Warning: number of bits of q does not
    (*Else*),
        securityParameters = <|"group"→pBits,"scalar"→qBits|>;
    ];
    If[securityParameters["group"] < securityPolicy["ElGamal"]["group"],VerificationFailed["S
    If[securityParameters["scalar"] < securityPolicy["ElGamal"]["scalar"],VerificationFailed[
    SetCostSpecification[securityParameters];
(*Else*),
    CheckSetup["EllipticCurve"];
    (*Set up and check Elliptic Curve*)
    If[encryptionMethod ≠ "EllipticCurve", Error["Setup Error: unknown encryption method "<>e
    curveRepresentation = setupParameters["crypt_curve_representation"];
    If[curveRepresentation ≠ "Weierstrass", Error["Setup Error: Unsupported Elliptic Curve re
    (*EC modulus*)
    p = setupParameters["crypt_p"];
    (*linear coefficient*)
```

```
    a_EC = setupParameters["crypt_a"];
    (*constant coefficient*)
    b_EC = setupParameters["crypt_b"];
    (*number of points on elliptic curve*)
    n_EC = setupParameters["crypt_n"];
    (*cofactor*)
    h_EC = setupParameters["crypt_h"];
    (*group modulus*)
    q = n_EC / h_EC;
    (*standard generator point*)
    g = setupParameters["crypt_g"];
    pBits = Length[IntegerDigits[p,2]];
    qBits = Length[IntegerDigits[q,2]];
    If[KeyExistsQ[setupParameters, "entropy_parameters"],
        entropyParameters = setupParameters["entropy_parameters"];
        If[pBits < entropyParameters["group"],Print["Warning: number of bits of p is smaller
        If[qBits < entropyParameters["scalar"],Print["Warning: number of bits of q is smaller
    (*Else*),
        entropyParameters = <|"group"→qBits,"scalar"→qBits|>;
    ];
    If[KeyExistsQ[setupParameters, "security_parameters"],
        securityParameters = setupParameters["security_parameters"];
        If[pBits + 1 (*sign bit*) ≠ securityParameters["group"],Print["Warning: Security para
        If[qBits ≠ securityParameters["scalar"],Print["Warning: Security parameters: Number o
    (*Else*),
        securityParameters = <|"group"→pBits+1(*sing bit*),"scalar"→qBits|>;
    ];
    setEC[ecCurve[a_EC,b_EC,{p,h_EC}]];
    compressPoint[point_ecPnt] := point[[1]] + If[point[[2]] > p/2, 2^pBits, 0];
    modq = Mod[#,q]&;
    modinvq = ModularInverse[#,q]&;
    powermodq = PowerMod[#1,#2,q]&;
    Print[Grid[{
         {"p is prime",PrimeQ[p]},
        {"bits of p",pBits},
        {"|n_EC - (p+1)| <= 2√n_EC ;\n(n_EC = number of points on EC)\n(Hasse's Theorem on ECs)",
        {"q is prime",PrimeQ[q]},
        {"bits of q",qBits},
        {"h_EC",h_EC},
        {"h_EC * q ≟ n_EC",h_EC * q == n_EC},
        {"qg = O",q * g == O}
    },Alignment→{{Left,Left}},Frame→All]];
    (*Selection of security and sanity checks*)
    If[!PrimeQ[p],VerificationFailed["Non-prime EC modulus p"]];
    If[!PrimeQ[q],VerificationFailed["Non-prime group order q"]];
    If[!IntegerQ[h_EC],VerificationFailed["Non-integer cofactor h_EC := n_EC/q"]];
    If[securityParameters["EC"]["group"] < securityPolicy["group"],VerificationFailed["Securi
    If[securityParameters["EC"]["scalar"] < securityPolicy["scalar"],VerificationFailed["Secu
    SetCostSpecification[securityParameters];
];

If[!interactive,
    setupParameters["nums_seed"] = fixedNUMSRandomSeed;
```

```
    Scan[
        FiatShamirAdd[
            #,
            If[StringQ[setupParameters[#]],
                StringToInteger[setupParameters[#]],
                setupParameters[#]
            ]
        ]&,
        (*sort to ensure fixed order*)
        Sort[Keys[setupParameters]]
    ];
];

Print[Grid[{
    {Style["Name/Equation",Bold],Style["Semantics",Bold],Style["Value/Equality",Bold]},
    {"n","Number of Elements\nto be Shuffled",n}
},Alignment→{{Left,Left,Left}},Frame→All]]

If[!interactive, Print[FiatShamirGrid[]]];
ProcessNewMessages[];
StateMachineSave["Setup"];
```

Waiting to receive setup from Prover ...

p is prime	True
bits of p	256
$\lvert n_{EC} - (p+1) \rvert <= 2\sqrt{n_{EC}}$; (n_{EC} = number of points on EC) (Hasse's Theorem on ECs)	True
q is prime	True
bits of q	256
h_{EC}	1
$h_{EC} * q \stackrel{?}{=} n_{EC}$	True
qg = $\mathcal{O}$	True

Name/Equation	Semantics	Value/Equality
n	Number of Elements to be Shuffled	3

NUMS Generators for Pedersen Commitments

"NUMS" ("nothing up my sleeve") generators for Pedersen Commitments. Created based on `fixedNUMSRandomSeed`

In[71]:=

```
StateMachineRestore["Setup"];

(*Standard "NUMS" generator point g is already given*)
(*Get more "NUMS" generator points with pseudorandom number generator initialized by fixed cor
(*Standard behaviour*)
BlockRandom[
    SeedRandom[fixedNUMSRandomSeed,Method→"ExtendedCA"];
    numsGenerators = ecRandomGenerators[p,h_EC,n+2];
];
h = numsGenerators⟦1⟧;
g⃗ = numsGenerators⟦2;;n+1⟧;
g' = numsGenerators⟦n+2⟧;
GiveInsight[{"h"→h,"gvec"→g⃗,"g'"→g'}];
Print[Grid[Join[
    {
        {Style["Name",Bold],Style["Knowledge",Bold],Style["Value",Bold]},
        {"g","public",g},
        {"h","public",h}
    },
    MapIndexed[{"g⃗."<>IntegerString[First[#2]],"public (implicit)",#1}&,g⃗],
    {
        {"g'","public",g'}
    }
], Alignment→{{Left,Left,Left}},Frame→All]]

If[!interactive,
    numsParameters = Join[
        <|
            (*Already added:
            "nums_seed"→fixedNUMSRandomSeed*)
            "nums_method"→StringToInteger["Mathematica-ExtendedCA"],
            "g"→compressPoint[g],
            "h"→compressPoint[h]
        |>,
        Association[Map["g."<>IntegerString[#]→compressPoint[g⃗⟦#⟧]&, Range[n]]],
        <|
            "g'"→compressPoint[g']
        |>
    ];
    Scan[
        FiatShamirAdd[#, numsParameters[#], 0]&,
        Sort[Keys[numsParameters]]
    ];
    Print[FiatShamirGrid[]];
]

ProcessNewMessages[];
StateMachineSave["NUMS"];
```

Name	Knowledge	Value
g	public	ecPnt[55 066 263 022 277 343 669 578 718 895 168 534 326 250 603 453 777 594 175 500 - 187 360 389 116 729 240, 32 670 510 020 758 816 978 083 085 130 507 043 184 471 273 380 659 243 275 938 - 904 335 757 337 482 424]
h	public	ecPnt[96 137 567 500 391 445 872 529 956 175 960 328 224 372 161 700 720 340 121 622 - 040 718 272 419 693 213, 70 732 012 083 304 338 620 554 688 440 613 834 674 667 959 557 579 739 145 719 - 658 468 136 375 884 866]
$\vec{g}$.1	public (implicit)	ecPnt[101 548 470 124 358 210 188 072 679 181 678 188 450 387 789 916 654 718 462 374 - 321 277 161 805 765 354, 70 108 350 633 035 199 759 065 376 354 370 719 210 357 510 544 392 766 796 027 - 711 414 587 086 150 112]
$\vec{g}$.2	public (implicit)	ecPnt[81 881 281 191 882 355 160 881 140 819 926 411 859 223 579 216 161 786 219 247 - 020 789 084 841 886 435, 40 551 526 063 420 644 421 580 965 386 725 729 859 544 147 053 123 598 525 382 - 531 556 934 666 119 757]
$\vec{g}$.3	public (implicit)	ecPnt[27 427 957 168 767 222 510 921 650 620 077 570 798 709 753 797 557 336 240 233 - 137 618 475 833 226 690, 42 650 441 509 520 284 448 005 737 515 710 331 630 163 209 846 762 957 134 645 - 512 392 724 356 208 176]
g'	public	ecPnt[56 472 720 079 496 714 458 128 130 368 877 680 293 647 814 114 642 090 837 679 - 403 063 618 918 620 862, 14 095 742 162 302 059 280 857 313 602 183 952 549 169 355 861 567 402 651 092 - 841 186 912 433 952 389]

Shuffle Proof

Get public $\vec{h}$, $\vec{b}$, $\vec{c}$, $\vec{d}$, $\vec{h}'$, $\vec{b}'$, $\vec{c}'$, $\vec{d}'$, Receive W and V

Public:

$\vec{h}$: public keys of receivers - to allow receivers to identify their messages

$\vec{b}$: commitments to blinding values - ElGamal randomizers

$\vec{c}$: blinded commitments to messages - ElGamal message

$\vec{d}$: public keys of senders - to allow receivers to reply

$\vec{h}'$: shuffled and shifted $\vec{h}$

$\vec{b}'$: shuffled and shifted $\vec{b}$

$\vec{c}'$: shuffled and shifted $\vec{c}$

$\vec{d}'$: shuffled and shifted $\vec{d}$

CRS: g, $\vec{g}$, $\vec{h}$, $\vec{b}$, $\vec{c}$, $\vec{d}$, $\vec{h}'$, $\vec{b}'$, $\vec{c}'$ and $\vec{d}'$

Commitment to Shift *s*:

V: commitment to s
W: commitment to s^n

In[81]:=
```
StateMachineRestore["NUMS"];

V = WaitForProversVariable["V"];
W = WaitForProversVariable["W"];

If[interactive,
(*now, that the verifier has received the encrypted shuffles, he knows the CRS is ready*)
    SendVariables["Inspector",{"__get_public_room_info__"→True,"__room_password__"→roomPassw
    roomInfo = WaitForVariable["__get_public_room_info__"];
    crs = roomInfo["crs"];
(*Else*),
(*Already received crs with previous request of public room info*)
    crs = roomInfo["crs"];
];

If[!KeyExistsQ[crs, "hvec"], VerificationFailed["Assertion Error: Missing 'hvec' in CRS"]];
If[!KeyExistsQ[crs, "bvec"], VerificationFailed["Assertion Error: Missing 'bvec' in CRS"]];
If[!KeyExistsQ[crs, "cvec"], VerificationFailed["Assertion Error: Missing 'cvec' in CRS"]];
If[!KeyExistsQ[crs, "dvec"], VerificationFailed["Assertion Error: Missing 'dvec' in CRS"]];
If[!KeyExistsQ[crs, "hprimevec"], VerificationFailed["Assertion Error: Missing 'hprimevec' in
If[!KeyExistsQ[crs, "bprimevec"], VerificationFailed["Assertion Error: Missing 'bprimevec' in
If[!KeyExistsQ[crs, "cprimevec"], VerificationFailed["Assertion Error: Missing 'cprimevec' in
If[!KeyExistsQ[crs, "dprimevec"], VerificationFailed["Assertion Error: Missing 'dprimevec' in
h⃗ = crs["hvec"];
b⃗ = crs["bvec"];
c⃗ = crs["cvec"];
d⃗ = crs["dvec"];
h⃗' = crs["hprimevec"];
b⃗' = crs["bprimevec"];
c⃗' = crs["cprimevec"];
d⃗' = crs["dprimevec"];

If[!interactive,
    Map[FiatShamirAdd["h."<>ToString[#], h⃗[[#]], 0]&, Range[n]];
    Map[FiatShamirAdd["b."<>ToString[#], b⃗[[#]], 0]&, Range[n]];
    Map[FiatShamirAdd["c."<>ToString[#], c⃗[[#]], 0]&, Range[n]];
    Map[FiatShamirAdd["d."<>ToString[#], d⃗[[#]], 0]&, Range[n]];
    Map[FiatShamirAdd["h'."<>ToString[#], h⃗'[[#]], 0]&, Range[n]];
    Map[FiatShamirAdd["b'."<>ToString[#], b⃗'[[#]], 0]&, Range[n]];
    Map[FiatShamirAdd["c'."<>ToString[#], c⃗'[[#]], 0]&, Range[n]];
    Map[FiatShamirAdd["d'."<>ToString[#], d⃗'[[#]], 0]&, Range[n]];
    FiatShamirAdd["V", compressPoint[V], entropyParameters["group"]];
    FiatShamirAdd["W", compressPoint[W], entropyParameters["group"]];
];
```

```
Print[Grid[{
    {Style["Name/Equation",Bold],Style["Semantics",Bold],Style["Value/Equality",Bold]},
    {"h⃗","Public Keys of Receivers for Message Identification","P,V",h⃗},
    {"b⃗","Commitments to γ⃗: ElGamal randomizer part","P,V",b⃗},
    {"c⃗","Blinded Commitments m⃗: ElGamal message part","P,V",c⃗},
    {"d⃗","Public Keys of Senders to Allow Receivers to Reply","P,V",d⃗},
    {"h⃗'","Shuffled and Shifted h⃗","P,V",h⃗'},
    {"b⃗'","Shuffled and Shifted b⃗","P,V",b⃗'},
    {"c⃗'","Shuffled and Shifted c⃗","P,V",c⃗'},
    {"d⃗'","Shuffled and Shifted d⃗","P,V",d⃗'},
    {"V","Commitment",V},
    {"W","Blinded Pedersen Commitment",W}
}, Alignment→{{Left,Left,Left}},Frame→All]]

If[!interactive, Print[FiatShamirGrid[]]];
ProcessNewMessages[];
StateMachineSave["Shuffle"];
```

```
Waiting to receive V from Prover ...

Waiting to receive W from Prover ...

Waiting to receive __get_public_room_info__ ...

Handling Event: success: __get_public_room_info__ (Inspector):
  Event Data: <|Length:7|>
  Keys: {status, __get_public_room_info__, _Message, _Valid, _SenderRoleAtId,
  _SenderRole, _SenderId, _ReceiverRoleAtId, _ReceiverRole, _ReceiverId,
  _UniqueKey, _Room, _DataFormat, _Data, _UnderscoreVars1, _UnderscoreVars2}
```

Name/Equation	Semantics	Value/Equality	
$\vec{h}$	Public Keys of Receivers for Message Identification	P,V	{ecPnt[78 567 409 798 129 814 866 - 150 056 060 307 810 399 - 673 849 671 094 211 427 - 553 792 618 009 791 659 - 223, 50 464 327 830 870 926 983 - 164 620 566 499 054 037 - 449 550 745 776 160 850 - 919 973 618 220 747 519 - 859], ecPnt[112 833 199 765 439 381 - 220 313 056 754 339 883 - 049 575 956 772 957 261 - 999 371 154 938 084 814 - 614 071, 55 162 480 627 138 884 082 - 362 925 002 107 050 797 - 199 473 890 589 593 047 - 995 087 693 795 006 543 - 339], ecPnt[112 833 199 765 439 381 - 220 313 056 754 339 883 - 049 575 956 772 957 261 - 999 371 154 938 084 814 - 614 071, 55 162 480 627 138 884 082 -

$\vec{b}$	Commitments to $\vec{\gamma}$: ElGamal randomizer part	P,V	{ecPnt[59 043 719 839 056 093 060 - 455 647 773 031 110 426 - 513 415 445 959 592 906 - 283 022 536 109 982 690 - 406, 15 581 121 562 794 375 473 - 593 189 690 062 526 490 - 521 359 132 176 103 829 - 752 471 918 632 370 295 - 336], ecPnt[28 042 937 753 915 930 788 - 547 013 599 645 448 755 - 092 123 652 825 491 664 - 072 915 209 989 352 569 - 125, 66 178 112 684 916 740 202 - 185 738 178 122 665 166 - 924 136 365 485 410 183 - 964 496 421 067 142 001 - 343], ecPnt[111 806 062 643 405 673 - 294 107 625 231 702 294 - 233 837 817 439 330 823 - 704 476 585 911 809 812 - 118 278, 96 271 173 562 246 442 335 - 118 096 000 002 956 980 - 355 600 440 300 789 883 - 776 848 991 926 176 478 - 624]}

$\vec{c}$	Blinded Commitments $\vec{m}$: ElGamal message part	P,V	{ecPnt[82 677 993 194 665 902 474 - 541 122 728 628 239 106 - 974 633 730 721 916 315 - 448 799 654 380 724 509 - 062, 82 899 414 855 368 741 404 - 724 030 068 539 514 612 - 899 114 324 925 700 999 - 638 050 403 382 505 421 - 116], ecPnt[23 172 912 405 278 733 415 - 435 710 602 144 815 479 - 916 588 170 014 597 418 - 408 278 116 825 280 333 - 989, 103 053 985 343 021 334 - 474 673 990 248 945 730 - 573 913 269 602 423 282 - 983 912 391 604 393 736 - 989 700], ecPnt[69 270 902 698 455 010 981 - 224 283 044 038 765 996 - 830 111 243 173 979 719 - 663 665 285 099 745 648 - 151, 112 380 697 802 250 453 - 283 363 092 945 878 117 - 421 176 770 345 454 476 - 832 123 069 113 820 053 - 308 875]}

$\vec{d}$	Public Keys of Senders to Allow Receivers to Reply	P,V	{ecPnt[97 038 873 997 667 556 164 - 890 627 159 143 816 584 - 036 737 601 412 047 181 - 786 115 325 137 586 601 - 893, 28 632 182 194 222 295 567 - 409 644 033 114 434 203 - 763 962 660 717 808 858 - 714 880 991 421 895 676 - 479], ecPnt[78 567 409 798 129 814 866 - 150 056 060 307 810 399 - 673 849 671 094 211 427 - 553 792 618 009 791 659 - 223, 50 464 327 830 870 926 983 - 164 620 566 499 054 037 - 449 550 745 776 160 850 - 919 973 618 220 747 519 - 859], ecPnt[53 729 729 513 994 595 695 - 462 455 195 591 489 616 - 567 069 832 745 850 818 - 442 954 791 457 454 586 - 503, 46 689 817 571 753 923 136 - 101 794 604 835 956 974 - 062 844 035 967 844 446 - 006 462 798 830 494 937 - 046]}

$\overrightarrow{h'}$	Shuffled and Shifted $\vec{h}$	P,V	{ecPnt[6 558 194 356 193 927 639 - 634 647 444 743 031 456 - 563 167 434 835 963 901 - 306 302 749 414 625 082 - 577, 52 488 910 954 347 611 664 - 776 036 389 515 076 273 - 016 905 927 124 685 240 - 883 874 979 762 544 524 - 251], ecPnt[6 558 194 356 193 927 639 - 634 647 444 743 031 456 - 563 167 434 835 963 901 - 306 302 749 414 625 082 - 577, 52 488 910 954 347 611 664 - 776 036 389 515 076 273 - 016 905 927 124 685 240 - 883 874 979 762 544 524 - 251], ecPnt[108 307 431 480 415 358 - 180 321 645 455 139 043 - 814 092 501 321 721 357 - 106 162 810 647 213 997 - 236 025, 36 351 342 789 842 377 099 - 456 363 842 641 744 188 - 644 647 315 961 192 968 - 880 873 781 984 072 160 - 774]}

$\overrightarrow{b'}$	Shuffled and Shifted $\vec{b}$	P,V	{ecPnt[2 101 504 872 291 888 883 - 716 359 192 847 722 039 - 905 370 848 787 468 118 - 146 221 520 733 335 251 - 689, 65 115 508 047 731 923 893 - 233 981 475 555 061 524 - 146 271 691 497 591 331 - 496 761 282 280 449 337 - 031], ecPnt[66 927 383 522 976 304 236 - 157 101 480 591 736 579 - 847 132 736 714 888 515 - 043 427 037 606 902 257 - 812, 107 572 068 642 340 469 - 124 979 747 235 810 229 - 109 194 492 671 695 182 - 627 325 466 548 702 521 - 834 811], ecPnt[54 597 720 163 964 310 354 - 822 001 511 722 288 403 - 820 425 105 673 732 255 - 729 486 794 940 621 160 - 858, 20 908 987 614 905 642 180 - 155 174 127 108 215 487 - 754 334 737 572 205 712 - 326 950 797 042 952 996 - 262]}

$\overrightarrow{c'}$	Shuffled and Shifted $\vec{c}$	P,V	{ecPnt[76 450 461 576 675 914 792 - 965 722 051 180 943 565 - 094 428 358 813 161 074 - 795 175 962 142 897 288 - 719, 38 216 996 469 148 933 963 - 905 268 396 466 886 302 - 699 197 112 751 046 297 - 550 831 186 589 191 968 - 048], ecPnt[33 216 804 232 678 278 471 - 260 719 141 100 536 638 - 697 095 655 969 466 196 - 649 583 487 480 597 031 - 229, 50 837 094 068 179 798 642 - 615 040 674 478 571 973 - 683 896 170 876 032 187 - 756 757 016 184 304 763 - 337], ecPnt[23 640 495 355 202 491 608 - 554 493 225 393 530 525 - 357 933 254 580 711 519 - 395 220 804 879 644 232 - 744, 63 745 865 814 030 357 294 - 809 620 337 129 418 176 - 053 657 581 482 167 229 - 204 651 942 724 506 248 - 630]}

$\overrightarrow{d'}$	Shuffled and Shifted $\vec{d}$	P,V	{ecPnt[108 307 431 480 415 358 - 180 321 645 455 139 043 - 814 092 501 321 721 357 - 106 162 810 647 213 997 - 236 025, 36 351 342 789 842 377 099 - 456 363 842 641 744 188 - 644 647 315 961 192 968 - 880 873 781 984 072 160 - 774], ecPnt[93 761 287 190 244 431 849 - 790 270 254 133 790 130 - 036 432 308 725 545 404 - 277 055 640 994 232 068 - 357, 35 500 574 849 318 626 019 - 937 037 834 919 246 834 - 175 404 043 884 801 123 - 182 806 451 365 357 886 - 461], ecPnt[112 929 678 132 582 868 - 571 013 728 088 079 645 - 858 007 540 176 146 226 - 385 430 663 184 270 576 - 277 149, 14 824 914 823 240 419 034 - 627 386 886 119 912 697 - 597 949 803 924 732 256 - 908 023 900 511 111 437 - 483]}
V	Commitment	ecPnt[80 311 418 295 773 201 369 - 076 106 032 905 885 135 - 762 203 375 069 519 910 - 440 352 607 757 637 919 - 026, 72 330 072 365 709 643 804 - 368 075 438 580 117 209 - 924 633 474 527 671 615 - 978 853 661 751 055 433 - 193]	
W	Blinded Pedersen Commitment	ecPnt[103 906 792 605 577 532 175 - 755 598 396 156 158 461 - 452 152 976 375 149 011 - 426 016 206 021 735 424 - 243, 86 390 539 398 352 229 318 - 624 618 945 653 159 494 - 571 754 482 845 117 036 - 288 422 074 734 839 991 - 103]	

Receive A_W and S_W for Verifying W being Well-Formed w.r.t. V

In addition to the correctness of key shift and shuffle it has to be proved that W which is a commitment to the n-th power of the key shift is well-formed with respect to V which is a commitment to the (first power of the) key shift. No details of this protocol are explained in the paper draft but it is nevertheless implemented alongside the shuffle argument in this notebook.
All variables exclusive to the PR argument contain a “W” in their names:
A_W, S_W, w, δ_{yzw}, $T_{1,W}$, $T_{2,W}$, $\tau_{x,W}$, $\hat{t_W}$, μ_W, P^+_W, $\overrightarrow{l_W}$, $\overrightarrow{r_W}$

In[106]:=

```
StateMachineRestore["Shuffle"];

Aw = WaitForProversVariable["AW"];
Sw = WaitForProversVariable["SW"];

If[!interactive,
    FiatShamirAdd["AW", compressPoint[Aw], entropyParameters["group"]];
    FiatShamirAdd["SW", compressPoint[Sw], entropyParameters["group"]];
];

Print[Grid[{
    {Style["Name/Equation",Bold],Style["Semantics",Bold],Style["Value/Equality",Bold]},
    {"Aw","Blinded\nPedersen\nVector\nCommitment",Aw},
    {"Sw","Blinded\nPedersen\nVector\nCommitment",Sw}
}, Alignment→{{Left,Left,Left}},Frame→All]]

Print[CommunicationCostGrid[]]
If[!interactive,Print[FiatShamirGrid[]]];
ProcessNewMessages[];
StateMachineSave["AWSW"];
```

```
Waiting to receive AW from Prover ...
```

```
Waiting to receive SW from Prover ...
```

Name/Equation	Semantics	Value/Equality
A_W	Blinded Pedersen Vector Commitment	ecPnt[102 533 074 015 578 785 830 435 668 723 656 382 032 639 089 329 076 188 012 - 258 139 630 558 261 897 965, 46 049 899 562 897 304 560 334 445 879 497 416 924 158 280 040 071 140 643 789 - 946 815 220 981 299 101]
S_W	Blinded Pedersen Vector Commitment	ecPnt[60 802 009 577 214 426 038 022 799 869 417 466 939 097 580 576 611 454 797 233 - 973 333 907 472 889 599, 4 544 274 886 182 003 749 351 257 221 639 993 502 991 294 730 296 803 633 157 - 620 175 479 609 245 008]

Total Communication Cost			
Type	Count	Names	Bits
group	4	V, W, AW, SW	1028
total	4		1028

Send Challenges r and u, calculate $\vec{k}$ and $A_{R,\{h,b,c,d\}}$

$k_i := r - u^i$ for all i in the range of 1 to n

$A_{R,\{h,b,c,d\}} := \Pi_{i=1}^{n} \left\{h_i', b_i', c_i', d_i'\right\}^{k_i}$

$A_{R,\{h,b,c,d\}}$ bind to the prover's secret $\overrightarrow{a_R}$ which is defined as the shifted and inversely shuffled version of $\vec{k}$ and therefore the list of discrete logarithms in $A_{R,\{h,b,c,d\}}$ w.r.t. bases $\left\{\vec{h}, \vec{b}, \vec{c}, \vec{d}\right\}$.

In[115]:=

```
StateMachineRestore["AWSW"];

If[interactive,
(*Generate random challenges*)
    r = SendRandomChallenge["r",1,q-1];
    u = SendRandomChallenge["u",1,q-1];
    fiatShamirCheck = {};
(*Else*),
(*Fiat-Shamir*)
    r = FiatShamirGet["r",q,entropyParameters["scalar"]];
    rProver = WaitForProversVariable["r"];
    fiatShamirCheckROk = (rProver == r);
    If[!fiatShamirCheckROk, GiveInsight[{"r"→r}]];
    u = FiatShamirGet["u",q,entropyParameters["scalar"]];
    uProver = WaitForProversVariable["u"];
    fiatShamirCheckUOk = (uProver == u);
    If[!fiatShamirCheckUOk, GiveInsight[{"u"→u}]];
    fiatShamirCheck = {
        {"Fiat-Shamir check on r:\nr_Prover ?= FiatShamir(setup,CRS,V,W,A_W,S_W)","",fiatShamirChecl
        {"Fiat-Shamir check on u:\nu_Prover ?= FiatShamir(r)","",fiatShamirCheckUOk}
    };
];

uPowers = Thread[powermodq[ConstantArray[u,n], Range[1,n]]];
k⃗ = modq[r - uPowers];
k̂ = modq[Fold[Times, 1, k⃗]];
k̄ = modq[Fold[Plus, 0, k⃗]];
A_R,h = Total[k⃗ * h⃗']
A_R,b = Total[k⃗ * b⃗']
A_R,c = Total[k⃗ * c⃗']
A_R,d = Total[k⃗ * d⃗']

Print[Grid[Join[
    {
        {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
        {"r",semanticsOfRandomChallenge,r},
        {"u",semanticsOfRandomChallenge,u}
    },
```

```
    fiatShamirCheck,
    {
        {"u_powers :≡_q {u,u^2,u^3,...,u^n}","",uPowers},
        {"k⃗ :≡_q r - u_powers","",k⃗},
        {"A_R,h := Σ{k⃗ ∘ h⃗'}","",A_R,h},
        {"A_R,b := Σ{k⃗ ∘ b⃗'}","",A_R,b},
        {"A_R,c := Σ{k⃗ ∘ c⃗'}","",A_R,c},
        {"A_R,d := Σ{k⃗ ∘ d⃗'}","",A_R,d}
    }
], Alignment→{{Left,Left,Left,Left}},Frame→All]]

Print[CommunicationCostGrid[]]
If[!interactive,Print[FiatShamirGrid[]]];
ProcessNewMessages[];
StateMachineSave["ruk"];

If[!interactive && (!fiatShamirCheckROk || !fiatShamirCheckUOk), VerificationFailed["Fiat-Sha
```

```
Generated random challenge
  r = 370724878441210165441995084583299037255046289629867413490589853230046420656 09
Generated random challenge
  u = 112269625660350362882863760160812690082016551740788795940136101880060177832694
```

Name	Semantics	Value
r	Random Challenge	37 072 487 844 121 016 544 199 508 458 329 903 725 504 628 - 962 986 741 349 058 985 323 004 642 065 609
u	Random Challenge	112 269 625 660 350 362 882 863 760 160 812 690 082 016 - 551 740 788 795 940 136 101 880 060 177 832 694
$u_{powers} :\equiv_q \{u, u^2, u^3, \ldots, u^n\}$		{112 269 625 660 350 362 882 863 760 160 812 690 082 016 - 551 740 788 795 940 136 101 880 060 177 832 694, 11 890 542 273 519 331 462 425 038 986 914 051 559 507 - 417 469 516 314 676 862 227 906 964 884 492 193, 33 387 804 159 995 661 087 204 252 238 941 081 331 379 - 259 117 507 535 049 098 238 738 582 016 269 690}
$\vec{k} :\equiv_q r - u_{powers}$		{40 594 951 421 086 849 084 906 733 306 205 121 496 325 - 641 501 272 849 791 528 046 584 462 625 727 252, 25 181 945 570 601 685 081 774 469 471 415 852 165 997 - 211 493 470 426 672 196 757 416 039 757 573 416, 3 684 683 684 125 355 456 995 256 219 388 822 394 125 369 - 845 479 206 299 960 746 584 422 625 795 919}
$A_{R,h} := \Sigma\{\vec{k} \circ \vec{h'}\}$		ecPnt[75 827 213 137 871 704 164 931 521 628 089 874 212 004 - 001 725 498 144 777 005 823 861 414 415 878 326, 85 558 088 274 753 361 702 239 458 714 109 752 170 257 - 343 893 657 124 488 281 320 372 885 377 844 970]
$A_{R,b} := \Sigma\{\vec{k} \circ \vec{b'}\}$		ecPnt[101 490 138 278 120 560 054 535 134 112 685 149 527 531 - 186 110 662 767 547 021 801 145 903 939 319 509, 45 189 396 226 369 639 764 475 183 750 210 875 191 161 - 366 914 100 613 038 683 080 011 606 878 795 742]
$A_{R,c} := \Sigma\{\vec{k} \circ \vec{c'}\}$		ecPnt[80 032 568 196 835 196 626 944 046 435 262 813 649 995 - 163 939 592 905 385 880 753 783 724 791 743 679, 23 552 969 371 290 033 158 266 674 238 835 159 465 211 - 446 720 669 348 929 745 850 171 431 031 988 187]
$A_{R,d} := \Sigma\{\vec{k} \circ \vec{d'}\}$		ecPnt[2 407 103 694 493 959 746 503 196 603 452 329 517 078 614 - 148 064 293 342 612 912 008 722 177 884 863, 56 880 823 889 132 509 467 030 174 309 379 436 312 737 - 498 520 799 651 845 161 746 864 517 931 461 934]

Total Communication Cost			
Type	**Count**	**Names**	**Bits**
group	4	V, W, AW, SW	1028
scalar	2	r, u	512
total	6		1540

Receive P's Commitments A_L, S_L, $S_{R,\{h,b,c,d\}}$

The commitments A_L, S_L and $S_{R,\{h,b,c,d\}}$ bind to $\vec{a_L}$, $\vec{s_L}$ and $\vec{s_R}$, the latter w.r.t. bases $\vec{h}$, $\vec{b}$, $\vec{c}$ and $\vec{d}$.

In[131]:=

```
StateMachineRestore["ruk"];

A_L = WaitForProversVariable["AL"];
If[!interactive,FiatShamirAdd["AL",compressPoint[A_L],entropyParameters["group"]]];

S_L = WaitForProversVariable["SL"];
If[!interactive,FiatShamirAdd["SL",compressPoint[S_L],entropyParameters["group"]]];

S_{R,h} = WaitForProversVariable["SRh"];
If[!interactive,FiatShamirAdd["SRh",compressPoint[S_{R,h}],entropyParameters["group"]]];

S_{R,b} = WaitForProversVariable["SRb"];
If[!interactive,FiatShamirAdd["SRb",compressPoint[S_{R,b}],entropyParameters["group"]]];

S_{R,c} = WaitForProversVariable["SRc"];
If[!interactive,FiatShamirAdd["SRc",compressPoint[S_{R,c}],entropyParameters["group"]]];

S_{R,d} = WaitForProversVariable["SRd"];
If[!interactive,FiatShamirAdd["SRd",compressPoint[S_{R,d}],entropyParameters["group"]]];

Print[Grid[{
    {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
    {"A_L","Blinded\nPedersen\nCommitment",A_L},
    {"S_L","Blinded\nPedersen\nCommitment",S_L},
    {"S_{R,h}","Blinded\nPedersen\nCommitment",S_{R,h}},
    {"S_{R,b}","Blinded\nPedersen\nCommitment",S_{R,b}},
    {"S_{R,c}","Blinded\nPedersen\nCommitment",S_{R,c}},
    {"S_{R,d}","Blinded\nPedersen\nCommitment",S_{R,d}}
}, Alignment→{{Left,Left,Left}},Frame→All]]

Print[CommunicationCostGrid[]]
If[!interactive,Print[FiatShamirGrid[]]];
ProcessNewMessages[];
StateMachineSave["AS"];
```

```
Waiting to receive AL from Prover ...
Waiting to receive SL from Prover ...
Waiting to receive SRh from Prover ...
Waiting to receive SRb from Prover ...
Waiting to receive SRc from Prover ...
Waiting to receive SRd from Prover ...
```

Name	Semantics	Value
A_L	Blinded Pedersen Commitment	ecPnt[6 587 839 308 200 477 344 056 289 937 852 477 545 542 855 779 583 120 343 135 425 480 544 - 699 054 079, 2 185 727 872 912 664 248 596 558 371 241 923 997 051 231 854 180 743 555 974 368 558 112 - 945 768 953]
S_L	Blinded Pedersen Commitment	ecPnt[4 034 094 399 684 424 483 307 446 142 353 323 528 168 707 067 153 775 234 243 263 723 285 - 535 027 265, 105 719 365 290 533 780 380 052 039 872 682 080 848 414 207 099 695 078 768 961 388 028 - 092 136 442 987]
$S_{R,h}$	Blinded Pedersen Commitment	ecPnt[110 628 460 759 275 561 610 186 686 680 845 830 158 167 592 932 204 791 477 090 968 094 - 876 973 217 838, 5 304 693 054 449 904 688 326 008 829 038 550 901 554 024 028 550 115 729 307 204 635 136 - 817 245 677]
$S_{R,b}$	Blinded Pedersen Commitment	ecPnt[42 640 815 911 507 716 472 477 938 340 233 126 446 598 231 831 337 676 705 110 847 057 - 191 691 627 240, 41 259 033 927 332 187 352 077 874 555 881 525 500 144 635 031 111 699 813 138 487 260 - 070 252 465 805]
$S_{R,c}$	Blinded Pedersen Commitment	ecPnt[62 635 959 701 511 398 581 525 732 065 492 888 015 851 399 229 301 995 870 753 638 223 - 458 110 077 747, 85 310 810 054 037 865 183 042 977 962 233 049 541 249 821 571 800 238 984 522 867 038 - 773 595 657 876]
$S_{R,d}$	Blinded Pedersen Commitment	ecPnt[93 333 551 283 426 889 209 650 746 890 228 923 405 712 607 594 824 962 743 876 131 424 - 188 627 252 440, 12 336 277 386 395 050 805 637 224 061 380 236 860 277 223 773 540 742 993 587 606 337 - 362 283 853 740]

Total Communication Cost			
Type	Count	Names	Bits
group	10	V, W, AW, SW, AL, SL, SRh, SRb, SRc, SRd	2570
scalar	2	r, u	512
total	12		3082

Send Random Challenges y and z or Generate Fiat-Shamir Transform

The random challenges y and z are used to enforce the linear constraints of the shuffle. For the PR protocol, an additional challenge w is needed.

In[149]:=

```
StateMachineRestore["AS"];
```

```
If[interactive,
(*Generate random challenges*)
    y = SendRandomChallenge["y",1,q-1];
    z = SendRandomChallenge["z",1,q-1];
    w = SendRandomChallenge["w",1,q-1];
    fiatShamirCheck = {};
(*Else*),
(*Fiat-Shamir*)
    y = FiatShamirGet["y",q,entropyParameters["scalar"]];
    yProver = WaitForProversVariable["y"];
    fiatShamirCheckYOk = (yProver == y);
    If[!fiatShamirCheckYOk, GiveInsight[{"y"→y}]];
    z = FiatShamirGet["z",q,entropyParameters["scalar"]];
    zProver = WaitForProversVariable["z"];
    fiatShamirCheckZOk = (zProver == z);
    If[!fiatShamirCheckZOk, GiveInsight[{"z"→z}]];
    w = FiatShamirGet["w",q,entropyParameters["scalar"]];
    wProver = WaitForProversVariable["w"];
    fiatShamirCheckWOk = (wProver == w);
    If[!fiatShamirCheckWOk, GiveInsight[{"w"→w}]];
    fiatShamirCheck = {
        {"Fiat-Shamir check on y:\ny_Prover ≟ FiatShamir(u,A_L,S_L,S_R,h,S_R,b,S_R,c)","",fiatShamirChec
        {"Fiat-Shamir check on z:\nz_Prover ≟ FiatShamir(y)","",fiatShamirCheckZOk},
        {"Fiat-Shamir check on w:\nw_Prover ≟ FiatShamir(z)","",fiatShamirCheckWOk}
    };
];

y⃗^n = Thread[powermodq[ConstantArray[y,n], Range[0,n-1]]];
y⃗_z^(n-1) = Join[{z},y⃗^n[[1;;n-1]]];
y^-1 = modinvq[y];
y⃗^-n = Thread[powermodq[ConstantArray[y^-1,n], Range[0,n-1]]];
w⃗^n = Thread[powermodq[ConstantArray[w,n], Range[0,n-1]]];
w⃗_0^(n-1) = Join[{0},w⃗^n[[1;;n-1]]];
δ_yz = modq[-z^3 * (z + (n-1) * y^-1) - z];
δ_yzw = modq[-z * (2 + z + (n-1) * y^-1) + (y^-1 - w) * (
    Total[Thread[powermodq[ConstantArray[w*y,n-1], Range[0,n-2]]]]
)];

Print[Grid[Join[
    {
        {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
        {"y",semanticsOfRandomChallenge,y},
        {"z",semanticsOfRandomChallenge,z},
        {"w",semanticsOfRandomChallenge,w}
    },
    fiatShamirCheck,
    {
        {"δ_yz :≡_q -z^3(z + (n-1)y^-1) - z","",δ_yz},
```

```
        {"δyzw :≡q -z (2 + z + (n-1)y^-1) + (y^-1-w)Σ_{i=0}^{n-2}(wy)^i","",δyzw}
    }
], Alignment→{{Left,Left,Left,Left}},Frame→All]]

Print[CommunicationCostGrid[]]
If[!interactive,Print[FiatShamirGrid[]]];
ProcessNewMessages[];
StateMachineSave["yzw"];

If[!interactive,
    If[!fiatShamirCheckYOk, VerificationFailed["Fiat-Shamir check on y"]];
    If[!fiatShamirCheckZOk, VerificationFailed["Fiat-Shamir check on z"]];
    If[!fiatShamirCheckWOk, VerificationFailed["Fiat-Shamir check on w"]];
];
```

Generated random challenge
  y = 340718071360204015565429663745208399664088018376676958871964006596574673562 50

Generated random challenge
  z = 512296503290655187206890960993798983120227599118044335244229717829425012994 78

Generated random challenge
  w = 427810565644108395011798979363110186512111534848283940100589304033010363902 38

Name	Semantics	Value
y	Random Challenge	34 071 807 136 020 401 556 542 966 374 520 - 839 966 408 801 837 667 695 887 196 400 - 659 657 467 356 250
z	Random Challenge	51 229 650 329 065 518 720 689 096 099 379 - 898 312 022 759 911 804 433 524 422 971 - 782 942 501 299 478
w	Random Challenge	42 781 056 564 410 839 501 179 897 936 311 - 018 651 211 153 484 828 394 010 058 930 - 403 301 036 390 238
$\delta_{yz} :\equiv_q -z^3(z + (n-1)y^{-1}) - z$		80 655 863 109 809 954 281 040 981 808 557 - 398 115 356 422 816 035 440 745 191 521 - 875 586 723 707 423
$\delta_{yzw} :\equiv_q -z\ (2 + z + (n-1)y^{-1}) + (y^{-1}-w)\sum_{i=0}^{n-2}(wy)^i$		36 280 791 154 615 824 673 741 371 562 371 - 573 961 477 989 289 960 370 376 293 582 - 689 521 148 309 524

Total Communication Cost			
Type	**Count**	**Names**	**Bits**
group	10	V, W, AW, SW, AL, SL, SRh, SRb, SRc, SRd	2570
scalar	5	r, u, y, z, w	1280
total	15		3850

Receive P's Pedersen Commitments T_3 and T_5, $T_{1,W}$ and $T_{2,W}$

In[165]:=

```
StateMachineRestore["yzw"];

(*Wait for Pedersen Commitment T₃ (to t₃ with blinding value τ₃)*)
T₃ = WaitForProversVariable["T3"];
If[!interactive,FiatShamirAdd["T3",compressPoint[T₃],entropyParameters["group"]]];

(*Wait for Pedersen Commitment T₅ (to t₅ with blinding value τ₅)*)
T₅ = WaitForProversVariable["T5"];
If[!interactive,FiatShamirAdd["T5",compressPoint[T₅],entropyParameters["group"]]];

Print[Grid[{
    {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
    {"T₃","Blinded\nPedersen\nCommitment",T₃},
    {"T₅","Blinded\nPedersen\nCommitment",T₅}
}, Alignment→{{Left,Left,Left}},Frame→All]]

(*Wait for Pedersen Commitment T₁,W (to t₁,W with blinding value τ₁,W for proving W is well-form
T₁,W = WaitForProversVariable["T1W"];
If[!interactive,FiatShamirAdd["T1W",compressPoint[T₁,W],entropyParameters["group"]]];

(*Wait for Pedersen Commitment T₂,W (to t₂,W with blinding value τ₂,W for proving W is well-form
T₂,W = WaitForProversVariable["T2W"];
If[!interactive,FiatShamirAdd["T2W",compressPoint[T₂,W],entropyParameters["group"]]];

Print[Grid[{
    {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
    {"T₁,W","Blinded\nPedersen\nCommitment",T₁,W},
    {"T₂,W","Blinded\nPedersen\nCommitment",T₂,W}
}, Alignment→{{Left,Left,Left}},Frame→All]]

Print[CommunicationCostGrid[]]
If[!interactive,Print[FiatShamirGrid[]]];
ProcessNewMessages[];
StateMachineSave["T3T5T1WT2W"];
```

Waiting to receive T3 from Prover ...

Waiting to receive T5 from Prover ...

Name	Semantics	Value
T_3	Blinded Pedersen Commitment	ecPnt[14 448 711 964 151 254 610 079 813 969 094 599 304 991 556 686 032 655 240 425 388 262 - 152 998 792 866, 35 818 652 155 586 706 410 279 698 238 883 774 802 777 588 417 348 300 197 902 141 088 - 629 232 808 578]
T_5	Blinded Pedersen Commitment	ecPnt[95 840 125 068 125 223 175 775 950 838 735 910 594 996 026 329 945 329 600 069 894 500 - 083 191 471 477, 92 410 008 359 066 548 507 274 553 560 973 459 366 308 871 128 888 129 891 539 618 716 - 611 325 817 962]

Waiting to receive T1W from Prover ...

Waiting to receive T2W from Prover ...

Name	Semantics	Value
$T_{1,W}$	Blinded Pedersen Commitment	ecPnt[5 443 615 670 032 252 682 449 032 168 926 607 403 789 720 879 129 525 399 520 688 082 435 - 286 679 707, 11 580 758 803 798 338 542 986 116 920 898 026 390 002 352 514 999 381 201 524 511 845 - 767 931 697 226]
$T_{2,W}$	Blinded Pedersen Commitment	ecPnt[46 991 716 425 756 598 968 559 577 543 620 785 423 160 955 041 139 713 706 964 410 377 - 968 981 965 216, 46 577 312 793 355 582 843 140 851 773 728 623 285 608 758 891 207 244 980 306 676 861 - 718 908 945 325]

Total Communication Cost			
Type	**Count**	**Names**	**Bits**
group	14	V, W, AW, SW, AL, SL, SRh, SRb, SRc, SRd, T3, T5, T1W, T2W	3598
scalar	5	r, u, y, z, w	1280
total	19		4878

Send Random Challenge x
or Generate Fiat-Shamir Transform

In[180]:=

```
StateMachineRestore["T3T5T1WT2W"];

(*Send random challenge x (evaluation point of t(X=x))*)
If[interactive,
(*Generate random challenge*)
    x = SendRandomChallenge["x",1,q-1];
    fiatShamirCheck = {};
(*Else*),
(*Fiat-Shamir*)
    x = FiatShamirGet["x",q,entropyParameters["scalar"]];
    xProver = WaitForProversVariable["x"];
    fiatShamirCheckXOk = (xProver == x);
    If[!fiatShamirCheckXOk, GiveInsight[{"x"→x}]];
    fiatShamirCheck = {
        {"Fiat-Shamir check on x:\nx_Prover ?= FiatShamir(setup,w,T_3,T_5,T_1,W,T_2,W)","",fiatShamirCh
    };

];

Print[Grid[Join[
    {
        {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
        {"x",semanticsOfRandomChallenge,x}
    },
    fiatShamirCheck
], Alignment→{{Left,Left,Left,Left}},Frame→All]]

Print[CommunicationCostGrid[]]
If[!interactive,Print[FiatShamirGrid[]]];
ProcessNewMessages[];
StateMachineSave["x"];

If[!interactive,
    If[!fiatShamirCheckXOk, VerificationFailed["Fiat-Shamir check on x"]];
];
```

Generated random challenge

x = 113126916120434927118052636603002490980047778302346487448691688890663263893727

Name	Semantics	Value
x	Random Challenge	113 126 916 120 434 927 118 052 636 603 002 490 980 047 778 302 346 487 448 691 688 890 663 263 893 727

Total Communication Cost			
Type	**Count**	**Names**	**Bits**
group	14	V, W, AW, SW, AL, SL, SRh, SRb, SRc, SRd, T3, T5, T1W, T2W	3598
scalar	6	r, u, y, z, w, x	1536
total	20		5134

Receive P's Combined Commitment to $t(X)$ and Perform Last Checks before Inner Product Protocol

Receiving

$\tau_x := \sigma_W \, \text{ny}^{n-1} \, \hat{k} \, x \, + \, \tau_3 \, x^3 \, + \, \tau_5 \, x^5$,

$\mu_{\{h,b,c,d\}} := \alpha + \beta x^2 + \rho_{\{h,b,c,d\}} \, x^3$ and

$\hat{t} := < \vec{l}, \vec{r} >$

from P where σ_W, τ_3, τ_5, α, β and $\rho_{\{h,b,c,d\}}$ are secret to P until inspection of his Pedersen Commitments. If the inner product protocol is not used, $\vec{l}$ and $\vec{r}$ are also received.

Also calculate $P^{+}_{\{h,b,c,d\}} := A_L + A_{R,\{h,b,c,d\}} \, x + S_L \, x^2 + S_{R,\{h,b,c,d\}} \, x^3 + z^3 < \vec{g}, \overrightarrow{y^{-n}} > + < \overrightarrow{\{h, b, c, d\}^*}, -x \, \overrightarrow{y_z^{n-1}} >$.

Perform the following checks:

1) In all cases:

$\hat{t} \, g + \tau_x \, h \stackrel{?}{=} z^3 \, \overline{k} \, \text{xV} + y^{n-1} \, \hat{k} \, \text{xW} + \delta_{\text{yz}} \, \text{xg} + x^3 \, T_3 + x^5 \, T_5$

Without the Inner Product Protocol:

2) $P^{+}_{\{h,b,c,d\}} \stackrel{?}{=} \mu_{\{h,b,c,d\}} \, h + < \vec{g}, \vec{l} > + < \overrightarrow{\{h, b, c, d\}^*}, \vec{r} >$

3) $\hat{t} \stackrel{?}{=} < \vec{l}, \vec{r} >$

When using the Inner Product Protocol 2) and 3) are proved indirectly.

In[188]:=

```
StateMachineRestore["x"];

(*Prover commits to t(x) and sends τx, μ{h,b,c,d} and t̂:*)
τx = WaitForProversVariable["taux"];
μh = WaitForProversVariable["muh"];
μb = WaitForProversVariable["mub"];
μc = WaitForProversVariable["muc"];
μd = WaitForProversVariable["mud"];
t̂ = WaitForProversVariable["t"];
If[!interactive,
    FiatShamirAdd["taux",τx,entropyParameters["scalar"]];
    FiatShamirAdd["muh",μh,entropyParameters["scalar"]];
    FiatShamirAdd["mub",μb,entropyParameters["scalar"]];
    FiatShamirAdd["muc",μc,entropyParameters["scalar"]];
    FiatShamirAdd["mud",μd,entropyParameters["scalar"]];
    FiatShamirAdd["t",t̂,entropyParameters["scalar"]];
];

h⃗* = y⁻ⁿ⃗*h⃗;
b⃗* = y⁻ⁿ⃗*b⃗;
c⃗* = y⁻ⁿ⃗*c⃗;
d⃗* = y⁻ⁿ⃗*d⃗;

Ph⁺ = AL + AR,h*x + SL*x^2 + SR,h*x^3 + g⃗.(z^3*y⁻ⁿ⃗) + h⃗*.(-x*y_z^(n-1)⃗);
```

```
P_b^+ = A_L + A_R,b*x + S_L*x^2 + S_R,b*x^3 + g⃗.(z^3*y^-n⃗) + b*⃗.(-x*y_z^n-1⃗);

P_c^+ = A_L + A_R,c*x + S_L*x^2 + S_R,c*x^3 + g⃗.(z^3*y^-n⃗) + c*⃗.(-x*y_z^n-1⃗);

P_d^+ = A_L + A_R,d*x + S_L*x^2 + S_R,d*x^3 + g⃗.(z^3*y^-n⃗) + d*⃗.(-x*y_z^n-1⃗);


(*1) t̂g + τ_x h ≟ z^3 k̄ x V + y^n-1 k̂ x W + δ_yz x g + x^3 T_3 + x^5 T_5*)
check1val1 = g*t̂ + h*τ_x;
check1val2 = V*(z^3*k̄*x) + W*(powermodq[y,n-1]*k̂*x) + g*(δ_yz*x) + T_3*x^3 + T_5*x^5;
check1 = (check1val1 == check1val2);

Print[Grid[{
    {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
    {"τ_x","Combined Commitment\nto Blinding Values\nof Commitments\nto Coefficiencts of t(X)",
    {"μ_h","Combined Commitment\nto Blinding Values\nof Vector Commitments\nto a⃗_… and s⃗_…\nBase
    {"μ_b","Combined Commitment\nto Blinding Values\nof Vector Commitments\nto a⃗_… and s⃗_…\nBase
    {"μ_c","Combined Commitment\nto Blinding Values\nof Vector Commitments\nto a⃗_… and s⃗_…\nBase
    {"μ_d","Combined Commitment\nto Blinding Values\nof Vector Commitments\nto a⃗_… and s⃗_…\nBase
    {"t̂","Inner Product",t̂},
    {"V_check1.1 := t̂g + τ_x h","",check1val1},
    {"V_check1.2 := z^3 k̄*V + k̂y^n-1 x*W + δ_yz x*g + x^3*T_3 + x^5*T_5","",check1val2},
    {"V_check1 := (V_check1.1 ≟ V_check1.2)",Style["Check No.1 for\nVerification of\nShuffle Proof ok?"
}, Alignment→{{Left,Left,Left}},Frame→All]]

(*Prover commits to t_W(x) and sends τ_x,W, μ_W and t̂_W:*)
τ_x,W = WaitForProversVariable["tauxW"];
μ_W = WaitForProversVariable["muW"];
t̂_W = WaitForProversVariable["tW"];
If[!interactive,
    FiatShamirAdd["tauxW",τ_x,W,entropyParameters["scalar"]];
    FiatShamirAdd["muW",μ_W,entropyParameters["scalar"]];
    FiatShamirAdd["tW",t̂_W,entropyParameters["scalar"]];
];

P_W^+ = A_W + S_W*x + g⃗.(z*y^-n⃗ + w^n⃗ - w_0^n-1⃗*y^-1) + h*⃗.(-y_z^n-1⃗);

(*1W) t̂_W g + τ_x,W h ≟ ((wy)^n-1+nz)V + y^n-1 W + δ_yzw g + x T_1,W + x^2 T_2,W*)
check1Wval1 = g*t̂_W + h*τ_x,W;
check1Wval2 = V*(powermodq[w*y,n-1]+n*z) + W*powermodq[y,n-1] + g*δ_yzw + T_1,W*x + T_2,W*x^2;
check1W = (check1Wval1 == check1Wval2);

Print[Grid[{
    {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
    {"τ_x,W","Combined Commitment\nto Blinding Values\nof Commitments\nto Coefficiencts of t_W(X)
    {"μ_W","Combined Commitment\nto Blinding Values\nof Vector Commitments\nto a⃗_…,W and s⃗_…,W",μ
```

```
    {"$\hat{t}_W$","Inner Product",$\hat{t}_W$},
    {"$V_{check1W.1}$ := $\hat{t}_W g + \tau_{x,W} h$","",check1Wval1},
    {"$V_{check1W.2}$ := $((wy)^{n-1}+nz)V + y^{n-1}W + \delta_{yzw} g + xT_{1,W} + x^2 T_{2,W}$","",check1Wval2},
    {"$V_{check1W}$ := ($V_{check1W.1} \stackrel{?}{=} V_{check1W.2}$)",Style["Check No.1W for\nVerification of\nShuffle Proof (
}, Alignment→{{Left,Left,Left}},Frame→All]]

Print[CommunicationCostGrid[]]
If[!interactive,Print[FiatShamirGrid[]]];
If[!check1, VerificationFailed["First check in multi-key shuffle argument failed"]];
If[!check1W, VerificationFailed["First check (PR) in multi-key shuffle argument failed"]];
ProcessNewMessages[];
StateMachineSave["mk_shuffle_argument_check"];

If[!useInnerProductProtocol,
    (*Need $\vec{l}$ and $\vec{r}$*)
    $\vec{l}$ = WaitForProversVariable["lvec"];
    $\vec{r}$ = WaitForProversVariable["rvec"];
    (*2) $P^+_{\{h,b,c,d\}} \stackrel{?}{=} h\mu_{\{h,b,c,d\}} + <\vec{g},\vec{l}> + <\overrightarrow{\{h,b,c,d\}^*},\vec{r}>$*)
    check2hval2 = $h*\mu_h + \vec{g}.\vec{l} + \overrightarrow{h^*}.\vec{r}$;
    check2h = ($P^+_h$ == check2hval2);
    check2bval2 = $h*\mu_b + \vec{g}.\vec{l} + \overrightarrow{b^*}.\vec{r}$;
    check2b = ($P^+_b$ == check2bval2);
    check2cval2 = $h*\mu_c + \vec{g}.\vec{l} + \overrightarrow{c^*}.\vec{r}$;
    check2c = ($P^+_c$ == check2cval2);
    check2dval2 = $h*\mu_d + \vec{g}.\vec{l} + \overrightarrow{d^*}.\vec{r}$;
    check2d = ($P^+_d$ == check2dval2);
    (*3) $t \stackrel{?}{=} <\vec{l},\vec{r}>$*)
    check3val2 = modq[$\vec{l}.\vec{r}$];
    check3 = ($\hat{t}$ == check3val2);
    (*PR: Need $\overrightarrow{l_W}$ and $\overrightarrow{r_W}$*)
    $\overrightarrow{l_W}$ = WaitForProversVariable["lWvec"];
    $\overrightarrow{r_W}$ = WaitForProversVariable["rWvec"];
    (*2W) $P^+_W \stackrel{?}{=} h\mu_W + <\vec{g},\overrightarrow{l_W}> + <\overrightarrow{h^*},\overrightarrow{r_W}>$*)
    check2Wval2 = $h*\mu_W + \vec{g}.\overrightarrow{l_W} + \overrightarrow{h^*}.\overrightarrow{r_W}$;
    check2W = ($P^+_W$ == check2Wval2);
    (*3W) $\hat{t}_W \stackrel{?}{=} <\overrightarrow{l_W},\overrightarrow{r_W}>$*)
    check3Wval2 = modq[$\overrightarrow{l_W}.\overrightarrow{r_W}$];
    check3W = ($\hat{t}_W$ == check3Wval2);
    Print[Grid[{
        {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
        {"$\vec{l}$","",$\vec{l}$},
        {"$\vec{r}$","",$\vec{r}$},
        {"$V_{check2h.2}$ := $h\mu_h + <\vec{g},\vec{l}> + <\overrightarrow{h^*},\vec{r}>$","",check2hval2},
        {"$V_{check2h}$ := ($P^+_h \stackrel{?}{=} V_{check2h.2}$)",Style["Check No.2h for\nVerification of\nShuffle Proof ok
```

```
        {"V_check2b.2 := hμ_b + <g⃗,l⃗> + <b⃗*,r⃗>","",check2bval2},
        {"V_check2b := (P_b^+ ≟ V_check2b.2)",Style["Check No.2b for\nVerification of\nShuffle Proof ok
        {"V_check2c.2 := hμ_c + <g⃗,l⃗> + <c⃗*,r⃗>","",check2cval2},
        {"V_check2c := (P_c^+ ≟ V_check2c.2)",Style["Check No.2c for\nVerification of\nShuffle Proof ok
        {"V_check2d.2 := hμ_d + <g⃗,l⃗> + <d⃗*,r⃗>","",check2dval2},
        {"V_check2d := (P_d^+ ≟ V_check2d.2)",Style["Check No.2d for\nVerification of\nShuffle Proof ok
        {"V_check3.2 :≡_q <l⃗,r⃗>","",check3val2},
        {"V_check3 := (t̂ ≟ V_check3.2)",Style["Check No.3 for\nVerification of\nShuffle Proof ok?",
        {"l⃗_W","",l⃗_W},
        {"r⃗_W","",r⃗_W},
        {"V_check2W.2 := hμ_W + <g⃗,l⃗_W> + <h⃗*,r⃗_W>","",check2Wval2},
        {"V_check2W := (P_W^+ ≟ V_check2W.2)",Style["Check No.2W for\nVerification of\nShuffle Proof ok
        {"V_check3W.2 :≡_q <l⃗_W,r⃗_W>","",check3Wval2},
        {"V_check3W := (t̂_W ≟ V_check3W.2)",Style["Check No.3W for\nVerification of\nShuffle Proof ok
    }, Alignment→{{Left,Left,Left}},Frame→All]];
    If[!check2h, VerificationFailed["Check No. 2h in multi-key shuffle argument failed"]];
    If[!check2b, VerificationFailed["Check No. 2b in multi-key shuffle argument failed"]];
    If[!check2c, VerificationFailed["Check No. 2c in multi-key shuffle argument failed"]];
    If[!check2d, VerificationFailed["Check No. 2d in multi-key shuffle argument failed"]];
    If[!check3, VerificationFailed["Check No. 3 in multi-key shuffle argument failed"]];
    If[!check2W, VerificationFailed["Check No. 2W in multi-key shuffle argument failed"]];
    If[!check3W, VerificationFailed["Check No. 3W in multi-key shuffle argument failed"]];
    VerificationSucceeded[];
    Print["Continuing to inner product protocol not required, but possible."];
    If[!interactive,Print[FiatShamirGrid[]]];
    Print[CommunicationCostGrid[]];
    ProcessNewMessages[];
    StateMachineSave["inner_product_argument_linear"];
];
```

```
Waiting to receive taux from Prover ...

Waiting to receive muh from Prover ...

Waiting to receive mub from Prover ...

Waiting to receive muc from Prover ...

Waiting to receive mud from Prover ...

Waiting to receive t from Prover ...
```

Name	Semantics	Value
τ_x	Combined Commitment to Blinding Values of Commitments to Coefficiencts of t(X)	90 664 101 823 773 446 617 843 295 664 - 000 476 824 583 008 189 854 797 930 - 932 933 141 663 523 670 972
μ_h	Combined Commitment to Blinding Values of Vector Commitments to $\overrightarrow{a_{...}}$ and $\overrightarrow{s_{...}}$ Based on $\vec{h}$	92 385 493 365 820 873 701 351 823 187 - 030 658 379 688 852 573 645 079 038 - 085 937 803 029 996 581 426
μ_b	Combined Commitment to Blinding Values of Vector Commitments to $\overrightarrow{a_{...}}$ and $\overrightarrow{s_{...}}$ Based on $\vec{b}$	80 465 903 251 238 947 246 948 019 231 - 634 112 958 643 085 797 558 256 913 - 390 431 732 924 637 611 796
μ_c	Combined Commitment to Blinding Values of Vector Commitments to $\overrightarrow{a_{...}}$ and $\overrightarrow{s_{...}}$ Based on $\vec{c}$	95 467 333 988 398 325 705 770 893 951 - 601 989 203 373 334 565 084 410 378 - 751 725 991 298 886 477 561
μ_d	Combined Commitment to Blinding Values of Vector Commitments to $\overrightarrow{a_{...}}$ and $\overrightarrow{s_{...}}$ Based on $\vec{d}$	20 751 440 986 732 760 450 071 315 784 - 341 311 844 859 651 372 502 581 770 - 822 695 204 710 647 943 339
$\hat{t}$	Inner Product	61 129 420 188 541 223 894 451 420 484 - 118 730 665 755 170 250 056 245 755 - 694 030 172 042 596 198 303
$V_{check1.1} := \hat{t}g + \tau_x h$		ecPnt[84 221 947 031 621 106 753 128 767 - 877 505 500 063 296 672 991 606 492 - 996 073 553 947 585 878 630 245, 52 570 167 137 227 745 050 217 253 - 766 965 056 131 545 854 005 981 366 - 395 868 066 499 760 183 049 215]
$V_{check1.2} := z^3\overline{k} * V + \hat{k}y^{n-1}x * W + \delta_{yz}x * g + x^3 * T_3 + x^5 * T_5$		ecPnt[84 221 947 031 621 106 753 128 767 - 877 505 500 063 296 672 991 606 492 - 996 073 553 947 585 878 630 245, 52 570 167 137 227 745 050 217 253 - 766 965 056 131 545 854 005 981 366 - 395 868 066 499 760 183 049 215]
$V_{check1} := (V_{check1.1} \stackrel{?}{=} V_{check1.2})$	**Check No.1 for Verification of Shuffle Proof ok?**	True

Waiting to receive tauxW from Prover ...

Waiting to receive muW from Prover ...

Waiting to receive tW from Prover ...

Name	Semantics	Value
$\tau_{x,W}$	Combined Commitment to Blinding Values of Commitments to Coefficiencts of $t_W(X)$	113 405 409 346 918 773 496 790 428 - 795 432 108 361 840 287 177 160 539 - 700 287 912 972 625 911 562 666
μ_W	Combined Commitment to Blinding Values of Vector Commitments to $\overrightarrow{a_{\ldots,W}}$ and $\overrightarrow{s_{\ldots,W}}$	58 938 274 415 372 746 259 906 244 - 789 991 376 153 504 860 263 629 609 - 061 215 650 209 757 990 735 313
$\hat{t}_W$	Inner Product	90 204 149 990 177 783 895 906 378 - 904 605 018 924 475 164 996 067 056 - 805 508 398 362 658 462 387 712
$V_{check1W.1} := \hat{t}_W g + \tau_{x,W} h$		ecPnt[71 170 761 461 806 575 056 988 777 - 799 149 753 588 821 351 113 719 - 544 888 720 840 260 093 735 125 151, 106 798 938 098 835 152 551 452 979 - 329 939 804 728 745 954 689 213 - 102 953 909 532 759 353 150 174 279]
$V_{check1W.2} := ((wy)^{n-1}+nz)V + y^{n-1}W + \delta_{yzw} g + xT_{1,W} + x^2 T_{2,W}$		ecPnt[71 170 761 461 806 575 056 988 777 - 799 149 753 588 821 351 113 719 - 544 888 720 840 260 093 735 125 151, 106 798 938 098 835 152 551 452 979 - 329 939 804 728 745 954 689 213 - 102 953 909 532 759 353 150 174 279]
$V_{check1W} := (V_{check1W.1} \stackrel{?}{=} V_{check1W.2})$	**Check No.1W for Verification of Shuffle Proof ok?**	True

Total Communication Cost			
Type	**Count**	**Names**	**Bits**
group	14	V, W, AW, SW, AL, SL, SRh, SRb, SRc, SRd, T3, T5, T1W, T2W	3598
scalar	15	r, u, y, z, w, x, taux, muh, mub, muc, mud, t, tauxW, muW, tW	3840
total	29		7438

Request Inspection of Pedersen Commitments (Multi-Key Shuffle and PR Protocol)

Request inspection of Pedersen commitments V, W, A_W, S_W, A_L, S_L, $S_{R,h}$, $S_{R,b}$, $S_{R,c}$, T_3, T_5, $T_{1,W}$, $T_{2,W}$ (multi-key shuffle protocol) as well as $L[n \ldots 1]$ and $R[n \ldots 1]$ (inner product protocol) from prover to verify that his commitments were done correctly. Additionally (but not necessary), check other variables exchanged during protocol execution.

In[224]:=

```
MultiKeyShuffleArgumentInspection[inspectionReply_] := Module[
    {success,
    AWInspectionOk,SWInspectionOk,
    hPrimePermutationOk,bPrimePermutationOk,cPrimePermutationOk,dPrimePermutationOk,
    VInspectionOk,WInspectionOk,
```

```
ALInspectionOk,SLInspectionOk,
SRhInspectionOk,SRbInspectionOk,SRcInspectionOk,SRdInspectionOk,
T3InspectionOk,t3DefinitionOk,
T5InspectionOk,t5DefinitionOk,
T1WInspectionOk,t1WDefinitionOk,
T2WInspectionOk,t2WDefinitionOk,
lDefinitionOk,rDefinitionOk,lrRow,
lWDefinitionOk,rWDefinitionOk,lWrWRow,
muhDefinitionOk,mubDefinitionOk,mucDefinitionOk,mudDefinitionOk,
tauxDefinitionOk,
tDefinitionOk,
tauxWDefinitionOk,
tWDefinitionOk},
s = inspectionReply["s"];
σW = inspectionReply["sigmaW"];
αW = inspectionReply["alphaW"];
βW = inspectionReply["betaW"];
γW = inspectionReply["gammaW"];
a⃗R,W = ConstantArray[s, n];
a⃗L,W = Thread[powermodq[a⃗R,W, Range[0,n-1]]];
s⃗R,W = ConstantArray[γW, n];
s⃗L,W = Thread[powermodq[ConstantArray[γW,n], Range[0,n-1]]];
AWInspectionOk = (AW == a⃗L,W.g⃗ + a⃗R,W.h⃗ + αW*h);
SWInspectionOk = (SW == s⃗L,W.g⃗ + s⃗R,W.h⃗ + βW*h);
Unprotect[π];
π = inspectionReply["pi"];
π^-1 = InversePermutation[π];
hPrimePermutationOk = (h⃗' == s * Permute[h⃗, π]);
bPrimePermutationOk = (b⃗' == s * Permute[b⃗, π]);
cPrimePermutationOk = (c⃗' == s * Permute[c⃗, π]);
dPrimePermutationOk = (d⃗' == s * Permute[d⃗, π]);
VInspectionOk = (V == s*g);
WInspectionOk = (W == powermodq[s,n]*g + σW*h);
a⃗R = modq[s * Permute[k⃗, π^-1]];
a⃗L = {1}; For[i = 1, i ≤ n-1, i++, AppendTo[a⃗L, modq[a⃗L[[i]] * a⃗R[[i]]]]];
α = inspectionReply["alpha"];
β = inspectionReply["beta"];
ρh = inspectionReply["rhoh"];
ρb = inspectionReply["rhob"];
ρc = inspectionReply["rhoc"];
ρd = inspectionReply["rhod"];
s⃗L = inspectionReply["sL"];
s⃗R = inspectionReply["sR"];
ALInspectionOk = (AL == a⃗L.g⃗ + α*h);
SLInspectionOk = (SL == s⃗L.g⃗ + β*h);
SRhInspectionOk = (SR,h == s⃗R.h⃗ + ρh*h);
SRbInspectionOk = (SR,b == s⃗R.b⃗ + ρb*h);
SRcInspectionOk = (SR,c == s⃗R.c⃗ + ρc*h);
```

```
SRdInspectionOk = (S_{R,d} == \vec{s_R}.\vec{d} + \rho_d*h);
\vec{l_0} = modq[\vec{a_L} + powermodq[z, 3] * \vec{y^{-n}}];
\vec{l_2} = \vec{s_L};
\vec{r_1} = modq[\vec{y^n} * \vec{a_R} - \vec{y_z^{n-1}}];
\vec{r_3} = modq[\vec{y^n} * \vec{s_R}];
t_3 = inspectionReply["t3"];
\tau_3 = inspectionReply["tau3"];
T3InspectionOk = (T_3 == t_3*g + \tau_3*h);
t3DefinitionOk = (modq[t_3] == modq[\vec{l_0}.\vec{r_3} + \vec{l_2}.\vec{r_1}]);
t_5 = inspectionReply["t5"];
\tau_5 = inspectionReply["tau5"];
T5InspectionOk = (T_5 == t_5*g + \tau_5*h);
t5DefinitionOk = (modq[t_5] == modq[\vec{l_2}.\vec{r_3}]);
\overrightarrow{l_{0,W}} = modq[\overrightarrow{a_{L,W}} + z*\vec{y^{-n}} + \vec{w^n} - \vec{w_0^{n-1}}*y^{-1}];
\overrightarrow{l_{1,W}} = \overrightarrow{s_{L,W}};
\overrightarrow{r_{0,W}} = modq[\overrightarrow{a_{R,W}}*\vec{y^n} - \vec{y_z^{n-1}}];
\overrightarrow{r_{1,W}} = modq[\overrightarrow{s_{R,W}}*\vec{y^n}];
t_{1,W} = inspectionReply["t1W"];
\tau_{1,W} = inspectionReply["tau1W"];
T1WInspectionOk = (T_{1,W} == t_{1,W}*g + \tau_{1,W}*h);
t1WDefinitionOk = (modq[t_{1,W}] == modq[\overrightarrow{l_{0,W}}.\overrightarrow{r_{1,W}} + \overrightarrow{l_{1,W}}.\overrightarrow{r_{0,W}}]);
t_{2,W} = inspectionReply["t2W"];
\tau_{2,W} = inspectionReply["tau2W"];
T2WInspectionOk = (T_{2,W} == t_{2,W}*g + \tau_{2,W}*h);
t2WDefinitionOk = (modq[t_{2,W}] == modq[\overrightarrow{l_{1,W}}.\overrightarrow{r_{1,W}}]);
If[useInnerProductProtocol,
    \vec{l} = modq[\vec{l_0} + \vec{l_2}*x^2];
    \vec{r} = modq[\vec{r_1}*x + \vec{r_3}*x^3];
    lDefinitionOk = True;
    rDefinitionOk = True;
    lrRow = {"\vec{l} :≡_q \vec{l_0}+\vec{l_2}x^2\n\vec{r} :≡_q \vec{r_1}x+\vec{r_3}x^3", SpanFromLeft, SpanFromLeft};
    \overrightarrow{l_W} = modq[\overrightarrow{l_{0,W}} + \overrightarrow{l_{1,W}}*x];
    \overrightarrow{r_W} = modq[\overrightarrow{r_{0,W}} + \overrightarrow{r_{1,W}}*x];
    \hat{t_W} = modq[\overrightarrow{l_W}.\overrightarrow{r_W}];
    lWDefinitionOk = True;
    rWDefinitionOk = True;
    lWrWRow = {"\overrightarrow{l_W} :≡_q \overrightarrow{l_{0,W}}+\overrightarrow{l_{1,W}}x\n\overrightarrow{r_W} :≡_q \overrightarrow{r_{0,W}}+\overrightarrow{r_{1,W}}x", SpanFromLeft, SpanFromLeft};
(*Else*),
    lDefinitionOk = (modq[\vec{l}] == modq[\vec{l_0} + \vec{l_2}*x^2]);
    rDefinitionOk = (modq[\vec{r}] == modq[\vec{r_1}*x + \vec{r_3}*x^3]);
    lrRow = {"\vec{l} ≟ \vec{l_0}+\vec{l_2}x^2\n\vec{r} ≟ \vec{r_1}x+\vec{r_3}x^3", "Definition\nDefinition", ToString[lDefinitionOk
    lWDefinitionOk = (modq[\overrightarrow{l_W}] == modq[\overrightarrow{l_{0,W}} + \overrightarrow{l_{1,W}}*x]);
```

```
    rWDefinitionOk = (modq[r⃗_W] == modq[r⃗_{0,W} + r⃗_{1,W}*x]);
    lWrWRow = {"l⃗_W ≟ l⃗_{0,W}+l⃗_{1,W}x\nr⃗_W ≟ r⃗_{0,W}+r⃗_{1,W}x", "Definition\nDefinition", ToString[lWDefi
];
muhDefinitionOk = (modq[μ_h] == modq[α + β*x^2 + ρ_h*x^3]);
mubDefinitionOk = (modq[μ_b] == modq[α + β*x^2 + ρ_b*x^3]);
mucDefinitionOk = (modq[μ_c] == modq[α + β*x^2 + ρ_c*x^3]);
mudDefinitionOk = (modq[μ_d] == modq[α + β*x^2 + ρ_d*x^3]);
tauxDefinitionOk = (modq[τ_x] == modq[σ_W*powermodq[y,n-1]*k̂*x + τ_3*x^3 + τ_5*x^5]);
tDefinitionOk = (modq[t̂] == modq[l⃗.r⃗]);
tauxWDefinitionOk = (modq[τ_{x,W}] == modq[σ_W*powermodq[y,n-1] + τ_{1,W}*x + τ_{2,W}*x^2]);
tWDefinitionOk = (modq[t̂_W] == modq[l⃗_W.r⃗_W]);
Print[Grid[{
    {Style["Expression",Bold], Style["Type of Check",Bold], Style["Value",Bold]},
    {"π", "Permutation", π},
    {"s", "Shift", s},
    {"A_W ≟ <a⃗_{L,W}, g⃗> + <a⃗_{R,W}, h⃗> + α_W h", "Inspection", AWInspectionOk},
    {"S_W ≟ <s⃗_{L,W}, g⃗> + <s⃗_{R,W}, h⃗> + β_W h", "Inspection", SWInspectionOk},
    {"h⃗' ≟ sπ(h⃗)", "Shifted permutation", hPrimePermutationOk},
    {"b⃗' ≟ sπ(b⃗)", "Shifted permutation", bPrimePermutationOk},
    {"c⃗' ≟ sπ(c⃗)", "Shifted permutation", cPrimePermutationOk},
    {"d⃗' ≟ sπ(d⃗)", "Shifted permutation", dPrimePermutationOk},
    {"V ≟ sg", "Inspection", VInspectionOk},
    {"W ≟ s^n g + σ_W h", "Inspection", WInspectionOk},
    {"a⃗_R :≡_q sπ^{-1}(k⃗)\na⃗_L :≡_q Multiplicative aggregation of a⃗_L starting at 1", SpanFromLeft,
    {"α", "", α},
    {"β", "", β},
    {"ρ_h", "", ρ_h},
    {"ρ_b", "", ρ_b},
    {"ρ_c", "", ρ_c},
    {"ρ_d", "", ρ_d},
    {"s⃗_L", "", s⃗_L},
    {"s⃗_R", "", s⃗_R},
    {"A_L ≟ <a⃗_L, g⃗> + αh", "Inspection", ALInspectionOk},
    {"S_L ≟ <s⃗_L, g⃗> + βh", "Inspection", SLInspectionOk},
    {"S_{R,h} ≟ <s⃗_R, h⃗> + ρ_h h", "Inspection", SRhInspectionOk},
    {"S_{R,b} ≟ <s⃗_R, b⃗> + ρ_b h", "Inspection", SRbInspectionOk},
    {"S_{R,c} ≟ <s⃗_R, c⃗> + ρ_c h", "Inspection", SRcInspectionOk},
    {"S_{R,d} ≟ <s⃗_R, d⃗> + ρ_d h", "Inspection", SRdInspectionOk},
    {"l⃗_0 :≡_q a⃗_L + z^3 y^{-n}⃗\nl⃗_2 :≡_q s⃗_L\nr⃗_1 :≡_q y^n⃗ ∘ a⃗_R - y_z^{n-1}⃗\nr⃗_3 :≡_q y^n⃗ ∘ s⃗_R", SpanFromLeft, SpanFro
    {"t_3", "", t_3},
    {"τ_3", "", τ_3},
```

```
    {"T_3 ≟ gt_3 + hτ_3", "Inspection", T3InspectionOk},
    {"t_3 ≟_q <l_0, r_3> + <l_2, r_1>", "Definition", t3DefinitionOk},
    {"t_5", "", t_5},
    {"τ_5", "", τ_5},
    {"T_5 ≟ gt_5 + hτ_5", "Inspection", T5InspectionOk},
    {"t_5 ≟_q <l_2, r_3>", "Definition", t5DefinitionOk},
    {"l_0,W :≡_q a_L,W + z y^-n + w^n - w_0^(n-1) y^-1 \nl_1,W :≡_q s_L,W \nr_0,W :≡_q y^n ∘ a_R,W - y_z^(n-1) \nr_1,W :≡_q y^n ∘ s_R,W
    {"t_1,W", "", t_1,W},
    {"τ_1,W", "", τ_1,W},
    {"T_1,W ≟ gt_1,W + hτ_1,W", "Inspection", T1WInspectionOk},
    {"t_1,W ≟_q <l_0,W, r_1,W> + <l_1,W, r_0,W>", "Definition", t1WDefinitionOk},
    {"t_2,W", "", t_2,W},
    {"τ_2,W", "", τ_2,W},
    {"T_2,W ≟ gt_2,W + hτ_2,W", "Inspection", T2WInspectionOk},
    {"t_2,W ≟_q <l_1,W, r_1,W>", "Definition", t2WDefinitionOk},
    lrRow,
    lWrWRow,
    {"μ_h ≟_q α + βx^2 + ρ_h x^3", "Definition", muhDefinitionOk},
    {"μ_b ≟_q α + βx^2 + ρ_b x^3", "Definition", mubDefinitionOk},
    {"μ_c ≟_q α + βx^2 + ρ_c x^3", "Definition", mucDefinitionOk},
    {"μ_d ≟_q α + βx^2 + ρ_d x^3", "Definition", mudDefinitionOk},
    {"τ_x ≟_q σ_W y^(n-1) k̂x + τ_3 x^3 + τ_5 x^5", "Definition", tauxDefinitionOk},
    {"t̂ ≟_q <l, r>", "Definition", tDefinitionOk},
    {"τ_x,W ≟_q σ_W y^(n-1) + τ_1,W x + τ_2,W x^2", "Definition", tauxWDefinitionOk},
    {"t̂_W ≟_q <l_W, r_W>", "Definition", tWDefinitionOk}
  }, Alignment→{{Left,Left,Left}},Frame→All]];

success = False;
Which[
    !AWInspectionOk, InspectionFailed["AW"],
    !SWInspectionOk, InspectionFailed["SW"],
    !hPrimePermutationOk, InspectionFailed["Permutation of h'"],
    !bPrimePermutationOk, InspectionFailed["Permutation of b'"],
    !cPrimePermutationOk, InspectionFailed["Permutation of c'"],
    !dPrimePermutationOk, InspectionFailed["Permutation of d'"],
    !VInspectionOk, InspectionFailed["V"],
    !WInspectionOk, InspectionFailed["W"],
    !ALInspectionOk, InspectionFailed["AL"],
    !SLInspectionOk, InspectionFailed["SL"],
    !SRhInspectionOk, InspectionFailed["SRh"],
    !SRbInspectionOk, InspectionFailed["SRb"],
    !SRcInspectionOk, InspectionFailed["SRc"],
    !SRdInspectionOk, InspectionFailed["SRd"],
```

```
        !T3InspectionOk, InspectionFailed["T3"],
        !t3DefinitionOk, InspectionFailed["Definition of t3"],
        !T5InspectionOk, InspectionFailed["T5"],
        !t5DefinitionOk, InspectionFailed["Definition of t5"],
        !T1WInspectionOk, InspectionFailed["T1W"],
        !t1WDefinitionOk, InspectionFailed["Definition of t1W"],
        !T2WInspectionOk, InspectionFailed["T2W"],
        !t2WDefinitionOk, InspectionFailed["Definition of t2W"],
        !lDefinitionOk, InspectionFailed["Definition of l vector"],
        !rDefinitionOk, InspectionFailed["Definition of r vector"],
        !lWDefinitionOk, InspectionFailed["Definition of lW vector"],
        !rWDefinitionOk, InspectionFailed["Definitino of lR vector"],
        !muhDefinitionOk, InspectionFailed["Definition of muh"],
        !mubDefinitionOk, InspectionFailed["Definition of mub"],
        !mucDefinitionOk, InspectionFailed["Definition of muc"],
        !mudDefinitionOk, InspectionFailed["Definition of mud"],
        !tauxDefinitionOk, InspectionFailed["Definition of taux"],
        !tDefinitionOk, InspectionFailed["Definition of t"],
        !tauxWDefinitionOk, InspectionFailed["Definition of tauxW"],
        !tWDefinitionOk, InspectionFailed["Definition of tW"],
        True, success = True
    ];
    success
]

If[!useInnerProductProtocol,
    If[fiatShamirSingleKernel,
        Print["Can not send inspection requests in Single-Kernel mode. Done."];
    (*Else*),
        StateMachineRestore["inner_product_argument_linear"];
        TellProver[{"_inspection_request_"→True}];
        inspectionReply = WaitForVariable["_inspection_reply","Prover"];
        (*Waiting for reply until timeout not possible in Mathematica:
        Not receiving an inspection reply in a certain amount of time
        should qualify as dishonest behaviour of the prover.*)
        If[MultiKeyShuffleArgumentInspection[inspectionReply],
            InspectionSucceeded[];
        ];
    ];
    FrontEndTokenExecute["EvaluatorAbort"];
];
```

Inner Product Protocol

Set up vectors and variables for successive halving

Pad $\vec{g}$, $\vec{h^*}$, $\vec{b^*}$, $\vec{c^*}$ and $\vec{d^*}$ such that their lengths become a power of two, resulting in the vectors $\vec{G}$, $\vec{H}$, $\vec{B}$, $\vec{C}$ and $\vec{D}$. Pad vectors of scalars with zeros and vectors of elliptic curve points with points at infinity. $\vec{b}$ *has a name collission with the randomizers of the ElGamal ciphertexts. Thus,* $\vec{a}$ *and* $\vec{b}$ *are indexed with "IP" in the code.*

Define $P'_{\{h,b,c,d\}} := A_L + A_{R,\{h,b,c,d\}}\, x + S_L\, x^2 + S_{R,\{h,b,c,d\}}\, x^3 + \langle \vec{g}, z^3 \overrightarrow{y^{-n}} \rangle + \langle \overrightarrow{\{h, b, c, d\}^*}, -x \overrightarrow{y_z^{n-1}} \rangle - \mu_{\{h,b,c,d\}}\, h + \hat{t}\, g'$
and $P'_W := A_W + S_W\, x + \langle \vec{g}, z \overrightarrow{y^{-n}} + \overrightarrow{w^n} - \overrightarrow{w_0^{n-1}} y^{-1} \rangle + \langle \overrightarrow{h^*}, -\overrightarrow{y_z^{n-1}} \rangle - \mu_W\, h + \hat{t_W}\, g'$.
$P'_{\{h,b,c,d\}}$ is equivalent to
$\langle \vec{a}, \vec{G} \rangle + \langle \vec{b}, \overrightarrow{\{H, B, C, D\}} \rangle + \langle \vec{a}, \vec{b} \rangle g' = \langle \vec{l}, \vec{g} \rangle + \langle \vec{r}, \overrightarrow{\{h, b, c, d\}^*} \rangle + \langle \vec{l}, \vec{r} \rangle g'$
and P'_W is equivalent to $\langle \overrightarrow{a_W}, \vec{G} \rangle + \langle \overrightarrow{b_W}, \vec{H} \rangle + \langle \overrightarrow{a_W}, \overrightarrow{b_W} \rangle g' = \langle \overrightarrow{l_W}, \vec{g} \rangle + \langle \overrightarrow{r_W}, \overrightarrow{h^*} \rangle + \langle \overrightarrow{l_W}, \overrightarrow{r_W} \rangle g'$ but only the prover can calculate $P'_{\{h,b,c,d,W\}}$ this way. Iteratively adding up those and all “halved” versions of $P'_{\{h,b,c,d,W\}}$ (which are yet to be calculated) is the central part of the inner product protocol which ensures the correctness of the inner product after verification.

In[226]:=

```
StateMachineRestore["mk_shuffle_argument_check"];

n_pow2 = 2^Ceiling[Log2[n]];
(*pad vectors*)
G⃗ = Join[g⃗[[1;;Min[{n_pow2,n}]]],ConstantArray[ecPnt[∞,∞],Max[0,n_pow2-n]]];
H⃗ = Join[h⃗*[[1;;Min[{n_pow2,n}]]],ConstantArray[ecPnt[∞,∞],Max[0,n_pow2-n]]];
B⃗ = Join[b⃗*[[1;;Min[{n_pow2,n}]]],ConstantArray[ecPnt[∞,∞],Max[0,n_pow2-n]]];
C⃗ = Join[c⃗*[[1;;Min[{n_pow2,n}]]],ConstantArray[ecPnt[∞,∞],Max[0,n_pow2-n]]];
D⃗ = Join[d⃗*[[1;;Min[{n_pow2,n}]]],ConstantArray[ecPnt[∞,∞],Max[0,n_pow2-n]]];

P_h' = P_h^+ - μ_h*h + t̂*g';
P_b' = P_b^+ - μ_b*h + t̂*g';
P_c' = P_c^+ - μ_c*h + t̂*g';
P_d' = P_d^+ - μ_d*h + t̂*g';
P_W' = P_W^+ - μ_W*h + t̂_W*g';

(*Keep track of different versions of G⃗, H⃗, B⃗, C⃗, D⃗, and P_{h,b,c,d,W} throughout halving steps*)
G⃗_2 = {G⃗};
H⃗_2 = {H⃗};
B⃗_2 = {B⃗};
C⃗_2 = {C⃗};
D⃗_2 = {D⃗};
P_2,h = {P_h'};
P_2,b = {P_b'};
P_2,c = {P_c'};
P_2,d = {P_d'};
P_2,W = {P_W'};

Print[Grid[{
    {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
    {"n_pow2 := 2^⌈log₂n⌉","Smallest power of 2\ngreater than\nor equal to n",n_pow2},
    {"G⃗ := padded g⃗","",G⃗},
```

```
    {"H⃗ := padded h⃗*","",H⃗},
    {"B⃗ := padded b⃗*","",B⃗},
    {"C⃗ := padded c⃗*","",C⃗},
    {"D⃗ := padded d⃗*","",D⃗},
    {"Ṕ_{h,b,c,d} ≟ P⁺_{h,b,c,d} - μ_{h,b,c,d} h + t̂g'","Combination of\nShuffle Proof\nConditions Based\n
    {"Ṕ_W ≟ P⁺_W - μ_W h + t̂_W g'","Combination of\nPR Proof Conditions",Ṕ_W}
}, Alignment→{{Left,Left,Left}},Frame→All]]

If[!interactive,Print[FiatShamirGrid[]]];
Print[CommunicationCostGrid[]];
ProcessNewMessages[];
StateMachineSave["inner_product_setup"];

If[!interactive && !fiatShamirCheckOk, VerificationFailed["Fiat-Shamir check on u'"]];
```

Name	Semantics	Value
$n_{pow2} := 2^{\lceil \log_2 n \rceil}$	Smallest power of 2 greater than or equal to n	4
$\vec{G} :=$ padded $\vec{g}$		{ecPnt[101 548 470 124 358 210 188 072 679 - 181 678 188 450 387 789 916 654 718 - 462 374 321 277 161 805 765 354, 70 108 350 633 035 199 759 065 376 354 - 370 719 210 357 510 544 392 766 796 - 027 711 414 587 086 150 112], ecPnt[81 881 281 191 882 355 160 881 140 819 - 926 411 859 223 579 216 161 786 219 - 247 020 789 084 841 886 435, 40 551 526 063 420 644 421 580 965 386 - 725 729 859 544 147 053 123 598 525 - 382 531 556 934 666 119 757], ecPnt[27 427 957 168 767 222 510 921 650 620 - 077 570 798 709 753 797 557 336 240 - 233 137 618 475 833 226 690, 42 650 441 509 520 284 448 005 737 515 - 710 331 630 163 209 846 762 957 134 - 645 512 392 724 356 208 176], ecPnt[∞, ∞]}

$\vec{H}$:= padded $\overrightarrow{h^*}$		{ecPnt[78 567 409 798 129 814 866 150 056 060 - 307 810 399 673 849 671 094 211 427 - 553 792 618 009 791 659 223, 50 464 327 830 870 926 983 164 620 566 - 499 054 037 449 550 745 776 160 850 - 919 973 618 220 747 519 859], ecPnt[16 230 325 873 944 567 770 651 181 130 - 303 324 912 121 224 490 802 407 282 - 022 789 649 993 284 143 768, 13 743 595 330 208 364 632 961 058 787 - 086 372 972 527 668 621 267 658 729 - 713 274 917 516 765 878 025], ecPnt[73 955 863 155 933 535 982 859 074 509 - 836 995 565 814 682 227 118 115 222 - 970 062 094 871 722 157 004, 12 661 142 626 516 128 143 965 107 877 - 524 977 948 748 772 063 737 046 982 - 621 034 957 911 954 836 350], ecPnt[∞, ∞]}
$\vec{B}$:= padded $\overrightarrow{b^*}$		{ecPnt[59 043 719 839 056 093 060 455 647 773 - 031 110 426 513 415 445 959 592 906 - 283 022 536 109 982 690 406, 15 581 121 562 794 375 473 593 189 690 - 062 526 490 521 359 132 176 103 829 - 752 471 918 632 370 295 336], ecPnt[112 521 091 029 611 710 525 094 888 - 851 919 040 284 401 236 803 289 858 - 290 779 025 976 942 166 966 892, 71 802 849 234 922 876 881 679 460 540 - 584 158 214 170 652 826 348 764 564 - 492 847 489 002 175 595 813], ecPnt[35 174 235 482 391 906 576 550 590 433 - 357 833 240 355 046 807 245 255 381 - 964 494 894 476 083 182 660, 55 925 870 782 800 122 155 770 079 253 - 820 122 879 523 420 329 479 534 761 - 202 990 107 031 888 939 945], ecPnt[∞, ∞]}

$\vec{C}$:= padded $\overrightarrow{c^*}$		{ecPnt[82 677 993 194 665 902 474 541 122 728 - 628 239 106 974 633 730 721 916 315 - 448 799 654 380 724 509 062, 82 899 414 855 368 741 404 724 030 068 - 539 514 612 899 114 324 925 700 999 - 638 050 403 382 505 421 116], ecPnt[45 422 098 385 527 520 105 729 242 155 - 385 912 239 396 352 275 187 691 869 - 915 510 620 991 934 660 246, 42 981 417 682 506 542 040 834 084 511 - 832 054 752 082 115 267 275 737 379 - 170 623 738 544 398 871 700], ecPnt[79 731 591 686 381 398 751 792 815 416 - 125 448 258 844 377 845 199 135 938 - 472 730 072 263 316 128 928, 14 986 293 206 980 309 680 514 267 840 - 081 157 662 963 426 467 477 088 209 - 824 552 014 914 871 589 040], ecPnt[∞, ∞]}
$\vec{D}$:= padded $\overrightarrow{d^*}$		{ecPnt[97 038 873 997 667 556 164 890 627 159 - 143 816 584 036 737 601 412 047 181 - 786 115 325 137 586 601 893, 28 632 182 194 222 295 567 409 644 033 - 114 434 203 763 962 660 717 808 858 - 714 880 991 421 895 676 479], ecPnt[114 943 785 603 625 565 834 113 287 - 895 479 632 817 105 457 530 959 156 - 178 552 924 525 977 788 156 149, 75 005 268 061 152 234 748 488 293 094 - 956 398 629 562 793 675 286 378 681 - 571 354 303 324 894 122 899], ecPnt[34 283 013 894 072 667 721 275 831 653 - 793 797 981 117 913 068 563 331 324 - 282 032 073 934 983 384 369, 16 444 072 431 226 081 049 726 942 532 - 624 576 021 955 849 773 018 079 843 - 576 838 182 070 874 161 038], ecPnt[∞, ∞]}

$P_{\{h,b,c,d\}} \stackrel{?}{=} P^{+}_{\{h,b,c,d\}} - \mu_{\{h,b,c,d\}} h + \hat{t} g'$	Combination of Shuffle Proof Conditions Based on $\vec{h}$, $\vec{b}$, $\vec{c}$ resp. $\vec{d}$	{ecPnt[42 427 373 215 793 781 659 408 924 843 - 768 227 225 882 770 225 375 757 126 - 397 503 527 068 891 674 844, 25 721 469 407 782 422 059 186 990 187 - 869 888 211 409 564 752 656 079 140 - 365 100 434 148 494 723 328], ecPnt[53 093 155 971 198 350 375 134 075 441 - 510 772 569 193 709 629 578 094 931 - 141 813 154 602 240 565 106, 55 938 320 630 024 527 613 951 665 036 - 283 624 008 160 716 735 992 177 830 - 942 127 603 092 847 901 156], ecPnt[87 623 278 363 384 013 501 616 299 287 - 548 132 476 250 177 902 650 041 648 - 463 953 140 336 403 353 190, 41 060 481 386 143 080 378 375 521 196 - 934 627 853 309 131 610 093 864 338 - 133 773 149 938 053 272 351], ecPnt[81 848 919 976 003 370 114 473 563 771 - 257 237 692 448 808 430 626 166 945 - 431 568 046 554 818 008 682, 2 643 826 270 596 107 626 712 310 049 - 127 795 608 650 776 351 598 726 801 - 898 843 301 631 070 212 794]}
$P_W \stackrel{?}{=} P^{+}_{W} - \mu_W h + \hat{t}_W g'$	Combination of PR Proof Conditions	ecPnt[98 991 610 698 448 567 601 040 979 656 - 886 525 408 488 940 163 120 548 218 - 252 896 669 433 046 079 564, 52 231 224 843 856 607 266 205 397 725 - 579 670 752 662 648 688 237 226 846 - 639 049 801 479 304 835 028]

Total Communication Cost			
Type	**Count**	**Names**	**Bits**
group	14	V, W, AW, SW, AL, SL, SRh, SRb, SRc, SRd, T3, T5, T1W, T2W	3598
scalar	15	r, u, y, z, w, x, taux, muh, mub, muc, mud, t, tauxW, muW, tW	3840
total	29		7438

Halving Iterations

Let $P_{\{h,b,c,W\}} := P'_{\{h,b,c,W\}}$

Continue to:

- Split and halve $\vec{G}$, $\vec{H}$, $\vec{B}$, and $\vec{C}$ based on challenge which is sent to prover,

- Receive commitments $L_{\{h,b,c,W\}}$ and $R_{\{h,b,c,W\}}$ to P's $\overrightarrow{a_{\text{IP}}}$ and $\overrightarrow{b_{\text{IP}}}$ resp. $\overrightarrow{a_W}$ and $\overrightarrow{b_W}$,

- Calculate halved versions of $P'_{\{h,b,c,W\}}$:

Split into lower and higher parts:

$\overrightarrow{G_{lo}} <> \overrightarrow{G_{hi}} := \vec{G}$

$\overrightarrow{H_{lo}} <> \overrightarrow{H_{hi}} := \vec{H}$

$\overrightarrow{B_{lo}} <> \overrightarrow{B_{hi}} := \vec{B}$

$\overrightarrow{C_{lo}} <> \overrightarrow{C_{hi}} := \vec{C}$

$\overrightarrow{D_{lo}} <> \overrightarrow{D_{hi}} := \vec{D}$

Receive commitments $L_{\{h,b,c,d,W\}}$ and $R_{\{h,b,c,d,W\}}$ from the prover.

Generate random challenge u_{IP} (called x in the Bulletproof paper) and send it to the prover. *To avoid name collission with previous challenge u index with "IP".*

Calculate halves:

$\overrightarrow{G_{next}} := u^{-1}\,\overrightarrow{G_{lo}} + u\,\overrightarrow{G_{hi}}$

$\overrightarrow{\{H,\,B,\,C,\,D\}_{next}} := u\,\overrightarrow{\{H,\,B,\,C,\,D\}_{lo}} + u^{-1}\,\overrightarrow{\{H,\,B,\,C,\,D\}_{hi}}$

$P_{next,\{h,b,c,d,W\}} := P_{\{h,b,c,d,W\}} + u^2\,L_{\{h,b,c,d,W\}} + u^{-2}\,R_{\{h,b,c,d,W\}}$

Set $P_{\{h,b,c,d,W\}} := P_{next,\{h,b,c,d,W\}}$ and go to next iteration until the size of the halved vectors has reached 1.

Last single-valued step:

$G_0 := \overrightarrow{G_{next}}[1]$

$H_0 := \overrightarrow{H_{next}}[1]$

$B_0 := \overrightarrow{B_{next}}[1]$

$C_0 := \overrightarrow{C_{next}}[1]$

$D_0 := \overrightarrow{D_{next}}[1]$

$P_{0,\{h,b,c,d,W\}} := P_{\{h,b,c,d,W\}}$

Receive a_0, b_0, $a_{0,W}$ and $b_{0,W}$ from the prover.

The "anatomy" of $P_{0,\{h,b,c,d,W\}}$ is: $P_{0,\{h,b,c,d,W\}} = P'_{\{h,b,c,d,W\}} + < u_2^2,\, L_{2,\{h,b,c,d,W\}} > + < u_2^{-2},\, R_{2,\{h,b,c,d,W\}} >$. This can be calculated as a sanity check at the end.

Calculate $c_0 := a_0\,b_0$ and check if $P_{0,\{h,b,c,d\}}$ equals $a_0\,G_0 + b_0\,\{H,\,B,\,C,\,D\}_0 + c_0\,g'$. Also check if $c_{0,W} := a_{0,W}\,b_{0,W}$ and if $P_{0,W}$ equals $a_{0,W}\,G_0 + b_{0,W}\,H_0 + c_{0,W}\,g'$.

These final verification steps mark the end of the inner product protocol.

Note on implementation: Indices of vectors count up from 1 to $\log_2 n_{pow2} + 1$ (`currentIteration`) and numbers for names of elements of vectors count down from $\log_2 n_{pow2}$ to 0 (`k, ciStr`), i.e. for $n_{pow2} = 16$: w[1]=w.4, w[2]=w.3, ... w[5]=w.0

In[254]:=

```
StateMachineRestore["inner_product_setup"];

n2 = {npow2};
k2 = {Log2[npow2]};

(*Re-initialize G, H, a, b and P in case of multiple cell evaluations*)
G2 = {G2[[1]]};
H2 = {H2[[1]]};
B2 = {B2[[1]]};
C2 = {C2[[1]]};
```

```
D2⃗ = {D2⃗[[1]]};
P2,h = {P2,h[[1]]};
P2,b = {P2,b[[1]]};
P2,c = {P2,c[[1]]};
P2,d = {P2,d[[1]]};
P2,W = {P2,W[[1]]};

L2,h = {};
L2,b = {};
L2,c = {};
L2,d = {};
L2,W = {};
R2,h = {};
R2,b = {};
R2,c = {};
R2,d = {};
R2,W = {};
u2 = {};

currentIteration = 1;

While[n2[[currentIteration]] ≠ 1,
    ciStr = IntegerString[k2[[currentIteration]]];
    Print[Grid[{
        {Style["Iteration "<>ciStr<>" (Counting from "<>IntegerString[k2[[1]]]<>" to 0)"], Spa
        {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
        {"k := k2."<>ciStr,"Current Iteration\n(counting down)",k2[[currentIteration]]},
        {"n2."<>ciStr<>" = 2^k","Current Length\nof Vectors",n2[[currentIteration]]},
        {"G⃗ := G2⃗."<>ciStr,"",G2⃗[[currentIteration]]},
        {"H⃗ := H2⃗."<>ciStr,"",H2⃗[[currentIteration]]},
        {"B⃗ := B2⃗."<>ciStr,"",B2⃗[[currentIteration]]},
        {"C⃗ := C2⃗."<>ciStr,"",C2⃗[[currentIteration]]},
        {"D⃗ := D2⃗."<>ciStr,"",D2⃗[[currentIteration]]},
        {"P{h,b,c,d,W} := P2,{h,b,c,d,W}."<>ciStr,"",{P2,h[[currentIteration]], P2,b[[currentIteration]],
    }, Alignment→{{Left,Left,Left}},Frame→All]];
    currentN = n2[[currentIteration]];
    halfN = currentN/2;
    k = k2[[currentIteration]];
    G⃗ = G2⃗[[currentIteration]];
    H⃗ = H2⃗[[currentIteration]];
    B⃗ = B2⃗[[currentIteration]];
    C⃗ = C2⃗[[currentIteration]];
    D⃗ = D2⃗[[currentIteration]];
    Ph = P2,h[[currentIteration]];
    Pb = P2,b[[currentIteration]];
    Pc = P2,c[[currentIteration]];
    Pd = P2,d[[currentIteration]];
    PW = P2,W[[currentIteration]];
```

```
(*Split into lower and higher parts*)
G⃗lo = G⃗[[1;;halfN]];
G⃗hi = G⃗[[halfN+1;;-1]];
H⃗lo = H⃗[[1;;halfN]];
H⃗hi = H⃗[[halfN+1;;-1]];
B⃗lo = B⃗[[1;;halfN]];
B⃗hi = B⃗[[halfN+1;;-1]];
C⃗lo = C⃗[[1;;halfN]];
C⃗hi = C⃗[[halfN+1;;-1]];
D⃗lo = D⃗[[1;;halfN]];
D⃗hi = D⃗[[halfN+1;;-1]];
(*Receive L and R from prover*)
Lh = WaitForProversVariable["Lh."<>ciStr];
Rh = WaitForProversVariable["Rh."<>ciStr];
Lb = WaitForProversVariable["Lb."<>ciStr];
Rb = WaitForProversVariable["Rb."<>ciStr];
Lc = WaitForProversVariable["Lc."<>ciStr];
Rc = WaitForProversVariable["Rc."<>ciStr];
Ld = WaitForProversVariable["Ld."<>ciStr];
Rd = WaitForProversVariable["Rd."<>ciStr];
LW = WaitForProversVariable["LW."<>ciStr];
RW = WaitForProversVariable["RW."<>ciStr];
AppendTo[L2,h,Lh];
AppendTo[R2,h,Rh];
AppendTo[L2,b,Lb];
AppendTo[R2,b,Rb];
AppendTo[L2,c,Lc];
AppendTo[R2,c,Rc];
AppendTo[L2,d,Ld];
AppendTo[R2,d,Rd];
AppendTo[L2,W,LW];
AppendTo[R2,W,RW];
If[!interactive,
    FiatShamirAdd["Lh."<>ciStr,compressPoint[Lh],entropyParameters["group"]];
    FiatShamirAdd["Rh."<>ciStr,compressPoint[Rh],entropyParameters["group"]];
    FiatShamirAdd["Lb."<>ciStr,compressPoint[Lb],entropyParameters["group"]];
    FiatShamirAdd["Rb."<>ciStr,compressPoint[Rb],entropyParameters["group"]];
    FiatShamirAdd["Lc."<>ciStr,compressPoint[Lc],entropyParameters["group"]];
    FiatShamirAdd["Rc."<>ciStr,compressPoint[Rc],entropyParameters["group"]];
    FiatShamirAdd["Ld."<>ciStr,compressPoint[Ld],entropyParameters["group"]];
    FiatShamirAdd["Rd."<>ciStr,compressPoint[Rd],entropyParameters["group"]];
    FiatShamirAdd["LW."<>ciStr,compressPoint[LW],entropyParameters["group"]];
    FiatShamirAdd["RW."<>ciStr,compressPoint[RW],entropyParameters["group"]];
];
Print[Grid[{
    {Style["Name",Bold],Style["Semantics",Bold],Style["Value",Bold]},
    {"L{h,b,c,d} := L2,{h,b,c,d}."<>ciStr,"Commitments to\n halves of a⃗ and b⃗\nbased on \n{h⃗,b⃗
    {"R{h,b,c,d} := R2,{h,b,c,d}."<>ciStr,"Commitments to\n other halves of a⃗ and b⃗\nbased on
    {"LW := L2,W."<>ciStr,"Commitment to\nhalves of\nā2,W⃗ and b̄2,W⃗\nfrom P"<>ciStr,L2,W[[curr
    {"RW := R2,W."<>ciStr,"Commitment to\nother halves of\nā2,W⃗ and b̄2,W⃗\nfrom P"<>ciStr,R2,
```

```
    }, Alignment→{{Left,Left,Left}},Frame→All]];
    If[interactive,
    (*Random challenge*)
        u_IP = SendRandomChallenge["u."<>ciStr,1,q-1];
        fiatShamirCheck = {};
    (*Else*),
    (*Fiat-Shamir*)
        u_IP = FiatShamirGet["u."<>ciStr,q,entropyParameters["scalar"]];
        uProver = WaitForProversVariable["u."<>ciStr];
        fiatShamirCheckOk = (uProver == u_IP);
        If[!fiatShamirCheckOk, GiveInsight[{"u."<>ciStr→u_IP}]];
        fiatShamirCheck = {
            {
                "Fiat-Shamir check on u."<>ciStr<>":\nc_Prover ≟ FiatShamir("<>
                    If[currentIteration==1,"u'","u."<>ToString[currentIteration-1]]<>
                    ",Lh."<>ciStr<>",Rh."<>ciStr<>
                    ",Lb."<>ciStr<>",Rb."<>ciStr<>
                    ",Lc."<>ciStr<>",Rc."<>ciStr<>
                    ",Ld."<>ciStr<>",Rd."<>ciStr<>
                    ",LW."<>ciStr<>",RW."<>ciStr<>")",
                "",
                fiatShamirCheckOk
            }
        };
    ];
    AppendTo[u_2,u_IP];
    u^-1 = modinvq[u_IP];
    G_next→ = u^-1*G_lo→+u_IP*G_hi→;
    AppendTo[G_2→,G_next→];
    H_next→ = u_IP*H_lo→+u^-1*H_hi→;
    AppendTo[H_2→,H_next→];
    B_next→ = u_IP*B_lo→+u^-1*B_hi→;
    AppendTo[B_2→,B_next→];
    C_next→ = u_IP*C_lo→+u^-1*C_hi→;
    AppendTo[C_2→,C_next→];
    D_next→ = u_IP*D_lo→+u^-1*D_hi→;
    AppendTo[D_2→,D_next→];
    P_next,h = P_h + u_IP^2*L_h + u^-1^2*R_h;
    AppendTo[P_2,h,P_next,h];
    P_next,b = P_b + u_IP^2*L_b + u^-1^2*R_b;
    AppendTo[P_2,b,P_next,b];
    P_next,c = P_c + u_IP^2*L_c + u^-1^2*R_c;
    AppendTo[P_2,c,P_next,c];
    P_next,d = P_d + u_IP^2*L_d + u^-1^2*R_d;
    AppendTo[P_2,d,P_next,d];
    P_next,W = P_W + u_IP^2*L_W + u^-1^2*R_W;
    AppendTo[P_2,W,P_next,W];
```

```
    Print[Grid[Join[
        {
            {Style["Fold G⃗, H⃗, B⃗, C⃗, D⃗ and P⃗ based on challenge u_IP"], SpanFromLeft},
            {Style["Name",Bold],Style["Value",Bold]},
            {"u_IP := u_2."<>ciStr,u_IP}
        },
        fiatShamirCheck,
        {
            {"G_next⃗ := u_IP^-1 G_lo⃗ + u_IP G_hi⃗", G_next⃗},
            {"H_next⃗ := u_IP H_lo⃗ + u_IP^-1 H_hi⃗", H_next⃗},
            {"B_next⃗ := u_IP B_lo⃗ + u_IP^-1 B_hi⃗", B_next⃗},
            {"C_next⃗ := u_IP C_lo⃗ + u_IP^-1 C_hi⃗", C_next⃗},
            {"D_next⃗ := u_IP D_lo⃗ + u_IP^-1 D_hi⃗", D_next⃗},
            {"P_next,{h,b,c,d,W} := P_{h,b,c,d,W} + u_IP^2 L_{h,b,c,d,W} + u_IP^-2 R_{h,b,c,d,W}", {P_next,h,P_next,b,P_next,c,P_ne
        }
    ], Alignment→{{Left,Left}},Frame→All]];
    (*Update variables for next iteration*)
    AppendTo[n_2,halfN];
    k = k - 1;
    AppendTo[k_2,k];
    currentIteration += 1;
]

(*Last, single-valued iteration*)
ciStr = IntegerString[k_2[[currentIteration]]];
n_0 = n_2[[currentIteration]];
k_0 = k_2[[currentIteration]];
G⃗ = G_2⃗[[currentIteration]];
H⃗ = H_2⃗[[currentIteration]];
B⃗ = B_2⃗[[currentIteration]];
C⃗ = C_2⃗[[currentIteration]];
D⃗ = D_2⃗[[currentIteration]];
P_h = P_2,h[[currentIteration]];
P_b = P_2,b[[currentIteration]];
P_c = P_2,c[[currentIteration]];
P_d = P_2,d[[currentIteration]];
P_W = P_2,W[[currentIteration]];
G_0 = G⃗[[1]];
H_0 = H⃗[[1]];
B_0 = B⃗[[1]];
C_0 = C⃗[[1]];
D_0 = D⃗[[1]];
P_0,h = P_h;
P_0,b = P_b;
P_0,c = P_c;
P_0,d = P_d;
```

```
P_{0,W} = P_W;

a_0 = WaitForProversVariable["a0"];
b_0 = WaitForProversVariable["b0"];
If[!interactive,
    FiatShamirAdd["a0",a_0,entropyParameters["scalar"]];
    FiatShamirAdd["b0",b_0,entropyParameters["scalar"]];
];
c_0 = modq[a_0*b_0];

a_{0,W} = WaitForProversVariable["a0W"];
b_{0,W} = WaitForProversVariable["b0W"];
If[!interactive,
    FiatShamirAdd["a0W",a_{0,W},entropyParameters["scalar"]];
    FiatShamirAdd["b0W",b_{0,W},entropyParameters["scalar"]];
];
c_{0,W} = modq[a_{0,W}*b_{0,W}];

P0hcheck = a_0*G_0 + b_0*H_0 + c_0*g';
P0bcheck = a_0*G_0 + b_0*B_0 + c_0*g';
P0ccheck = a_0*G_0 + b_0*C_0 + c_0*g';
P0dcheck = a_0*G_0 + b_0*D_0 + c_0*g';
P0Wcheck = a_{0,W}*G_0 + b_{0,W}*H_0 + c_{0,W}*g';

Print[Grid[{
    {Style["Iteration "<>ciStr<>" (Counting from "<>IntegerString[k_2[[1]]]<>" to 0)"], SpanFrom
    {Style["Name",Bold],Style["Semantics"],Style["Value",Bold]},
    {"n_0 := n_2."<>ciStr<>"[1] ?= 1","Halving Complete?",n_0==1},
    {"k_0 := k_2."<>ciStr<>"[1] ?= 0","Halving Complete?","P,V",k_0==0},
    {"G_0 := G_2^→."<>ciStr<>"[1]","",G_0},
    {"H_0 := H_2^→."<>ciStr<>"[1]","",H_0},
    {"B_0 := B_2^→."<>ciStr<>"[1]","",B_0},
    {"C_0 := C_2^→."<>ciStr<>"[1]","",C_0},
    {"D_0 := D_2^→."<>ciStr<>"[1]","",D_0},
    {"a_0","",a_0},
    {"b_0","",b_0},
    {"a_{0,W}","",a_{0,W}},
    {"b_{0,W}","",b_{0,W}},
    {"P_{0,{h,b,c,d,W}} := P_{2,{h,b,c,d,W}}."<>ciStr,"",{P_{0,h}, P_{0,b}, P_{0,c}, P_{0,d}, P_{0,W}}},
    {"c_0 := a_0b_0","",c_0},
    {"c_{0,W} := a_{0,W}b_{0,W}","",c_{0,W}},
    {"P_{0,{h,b,c,d,W}} ?= P'_{{h,b,c,d,W}} + <u_2^2,L_{2,{h,b,c,d,W}}> + <u_2^{-2},R_{2,{h,b,c,d,W}}>","Sanity Check\nfor P_0",{P_
    {"P_{0,{h,b,c,d}} ?= G_0a_0 + {H,B,C,D}_0b_0 + c_0g'",Style["Check No.2{h,b,c,d} for\nVerification of\
    {"P_{0,W} ?= G_0a_{0,W} + H_0b_{0,W} + c_{0,W}g'",Style["Check No.2W for\nVerification of\nPR Proof\n(with I
}, Alignment→{{Left,Left,Left}},Frame→All]]

Print["Inner product protocol finished."]
```

```
Print[CommunicationCostGrid[]]
If[!interactive,Print[FiatShamirGrid[]]];
ProcessNewMessages[];
StateMachineSave["inner_product_halving"];

If[P0,h≠P0hcheck,
    VerificationFailed["Check No.2h for Verification of Range Proof (with Inner Product Protoc
(*Else*),
If[P0,b≠P0bcheck,
    VerificationFailed["Check No.2b for Verification of Range Proof (with Inner Product Protoc
(*Else*),
If[P0,c≠P0ccheck,
    VerificationFailed["Check No.2c for Verification of Range Proof (with Inner Product Protoc
(*Else*),
If[P0,d≠P0dcheck,
    VerificationFailed["Check No.2d for Verification of Range Proof (with Inner Product Protoc
(*Else*),
If[P0,W≠P0Wcheck,
    VerificationFailed["Check No.2W for Verification of Range Proof (with Inner Product Protoc
(*Else*),
    VerificationSucceeded[];
]]]]]
```

Iteration 2 (Counting from 2 to 0)

Name	Semantics	Value
$k := k_2.2$	Current Iteration (counting down)	2
$n_2.2 = 2^k$	Current Length of Vectors	4
$\vec{G} := \vec{G_2}.2$		{ecPnt[101 548 470 124 358 210 188 072 679 181 678 188 450 - 387 789 916 654 718 462 374 321 277 161 805 765 - 354, 70 108 350 633 035 199 759 065 376 354 370 719 210 - 357 510 544 392 766 796 027 711 414 587 086 150 - 112], ecPnt[81 881 281 191 882 355 160 881 140 819 926 411 859 - 223 579 216 161 786 219 247 020 789 084 841 886 - 435, 40 551 526 063 420 644 421 580 965 386 725 729 859 - 544 147 053 123 598 525 382 531 556 934 666 119 - 757], ecPnt[27 427 957 168 767 222 510 921 650 620 077 570 798 - 709 753 797 557 336 240 233 137 618 475 833 226 - 690, 42 650 441 509 520 284 448 005 737 515 710 331 630 - 163 209 846 762 957 134 645 512 392 724 356 208 - 176], ecPnt[∞, ∞]}

$\vec{H} := \overrightarrow{H_2}.2$		{ecPnt[78 567 409 798 129 814 866 150 056 060 307 810 399 - 673 849 671 094 211 427 553 792 618 009 791 659 - 223, 50 464 327 830 870 926 983 164 620 566 499 054 037 - 449 550 745 776 160 850 919 973 618 220 747 519 - 859], ecPnt[16 230 325 873 944 567 770 651 181 130 303 324 912 - 121 224 490 802 407 282 022 789 649 993 284 143 - 768, 13 743 595 330 208 364 632 961 058 787 086 372 972 - 527 668 621 267 658 729 713 274 917 516 765 878 - 025], ecPnt[73 955 863 155 933 535 982 859 074 509 836 995 565 - 814 682 227 118 115 222 970 062 094 871 722 157 - 004, 12 661 142 626 516 128 143 965 107 877 524 977 948 - 748 772 063 737 046 982 621 034 957 911 954 836 - 350], ecPnt[∞, ∞]}
$\vec{B} := \overrightarrow{B_2}.2$		{ecPnt[59 043 719 839 056 093 060 455 647 773 031 110 426 - 513 415 445 959 592 906 283 022 536 109 982 690 - 406, 15 581 121 562 794 375 473 593 189 690 062 526 490 - 521 359 132 176 103 829 752 471 918 632 370 295 - 336], ecPnt[112 521 091 029 611 710 525 094 888 851 919 040 284 - 401 236 803 289 858 290 779 025 976 942 166 966 - 892, 71 802 849 234 922 876 881 679 460 540 584 158 214 - 170 652 826 348 764 564 492 847 489 002 175 595 - 813], ecPnt[35 174 235 482 391 906 576 550 590 433 357 833 240 - 355 046 807 245 255 381 964 494 894 476 083 182 - 660, 55 925 870 782 800 122 155 770 079 253 820 122 879 - 523 420 329 479 534 761 202 990 107 031 888 939 - 945], ecPnt[∞, ∞]}

$\vec{C}$:= $\overrightarrow{C_2}$.2		{ecPnt[82 677 993 194 665 902 474 541 122 728 628 239 106 - 974 633 730 721 916 315 448 799 654 380 724 509 - 062, 82 899 414 855 368 741 404 724 030 068 539 514 612 - 899 114 324 925 700 999 638 050 403 382 505 421 - 116], ecPnt[45 422 098 385 527 520 105 729 242 155 385 912 239 - 396 352 275 187 691 869 915 510 620 991 934 660 - 246, 42 981 417 682 506 542 040 834 084 511 832 054 752 - 082 115 267 275 737 379 170 623 738 544 398 871 - 700], ecPnt[79 731 591 686 381 398 751 792 815 416 125 448 258 - 844 377 845 199 135 938 472 730 072 263 316 128 - 928, 14 986 293 206 980 309 680 514 267 840 081 157 662 - 963 426 467 477 088 209 824 552 014 914 871 589 - 040], ecPnt[∞, ∞]}
$\vec{D}$:= $\overrightarrow{D_2}$.2		{ecPnt[97 038 873 997 667 556 164 890 627 159 143 816 584 - 036 737 601 412 047 181 786 115 325 137 586 601 - 893, 28 632 182 194 222 295 567 409 644 033 114 434 203 - 763 962 660 717 808 858 714 880 991 421 895 676 - 479], ecPnt[114 943 785 603 625 565 834 113 287 895 479 632 817 - 105 457 530 959 156 178 552 924 525 977 788 156 - 149, 75 005 268 061 152 234 748 488 293 094 956 398 629 - 562 793 675 286 378 681 571 354 303 324 894 122 - 899], ecPnt[34 283 013 894 072 667 721 275 831 653 793 797 981 - 117 913 068 563 331 324 282 032 073 934 983 384 - 369, 16 444 072 431 226 081 049 726 942 532 624 576 021 - 955 849 773 018 079 843 576 838 182 070 874 161 - 038], ecPnt[∞, ∞]}

$P_{\{h,b,c,d,W\}} := P_{2,\{h,b,c,d,W\}}.2$		{ecPnt[42 427 373 215 793 781 659 408 924 843 768 227 225 - 882 770 225 375 757 126 397 503 527 068 891 674 - 844, 25 721 469 407 782 422 059 186 990 187 869 888 211 - 409 564 752 656 079 140 365 100 434 148 494 723 - 328], ecPnt[53 093 155 971 198 350 375 134 075 441 510 772 569 - 193 709 629 578 094 931 141 813 154 602 240 565 - 106, 55 938 320 630 024 527 613 951 665 036 283 624 008 - 160 716 735 992 177 830 942 127 603 092 847 901 - 156], ecPnt[87 623 278 363 384 013 501 616 299 287 548 132 476 - 250 177 902 650 041 648 463 953 140 336 403 353 - 190, 41 060 481 386 143 080 378 375 521 196 934 627 853 - 309 131 610 093 864 338 133 773 149 938 053 272 - 351], ecPnt[81 848 919 976 003 370 114 473 563 771 257 237 692 - 448 808 430 626 166 945 431 568 046 554 818 008 - 682, 2 643 826 270 596 107 626 712 310 049 127 795 608 - 650 776 351 598 726 801 898 843 301 631 070 212 - 794], ecPnt[98 991 610 698 448 567 601 040 979 656 886 525 408 - 488 940 163 120 548 218 252 896 669 433 046 079 - 564, 52 231 224 843 856 607 266 205 397 725 579 670 752 - 662 648 688 237 226 846 639 049 801 479 304 835 - 028]}

Waiting to receive Lh.2 from Prover ...

Waiting to receive Rh.2 from Prover ...

Waiting to receive Lb.2 from Prover ...

Waiting to receive Rb.2 from Prover ...

Waiting to receive Lc.2 from Prover ...

Waiting to receive Rc.2 from Prover ...

Waiting to receive Ld.2 from Prover ...

Waiting to receive Rd.2 from Prover ...

Waiting to receive LW.2 from Prover ...

Waiting to receive RW.2 from Prover ...

Name	Semantics	Value
$L_{\{h,b,c,d\}} := L_{2,\{h,b,c,d\}}.2$	Commitments to halves of $\vec{a}$ and $\vec{b}$ based on $\{\vec{h},\vec{b},\vec{c},\vec{d}\}$ from P	{ecPnt[51 937 848 323 061 722 610 475 181 980 830 591 - 914 660 958 406 428 433 789 558 240 107 198 - 247 801 268, 106 560 155 938 380 029 201 930 354 580 040 167 - 551 432 141 775 760 085 148 764 943 146 178 - 634 187 408], ecPnt[79 863 622 949 771 475 243 779 360 474 452 590 - 639 328 325 896 457 338 087 659 196 341 407 - 241 191 284, 105 853 781 236 695 666 215 411 488 447 363 145 - 637 561 962 496 056 988 894 367 134 599 023 - 782 061 088], ecPnt[63 735 917 209 878 754 899 423 202 716 828 621 -

$R_{\{h,b,c,d\}} := R_{2,\{h,b,c,d\}}.2$	Commitments to other halves of $\vec{a}$ and $\vec{b}$ based on $\{\vec{h},\vec{b},\vec{c},\vec{d}\}$ from P	{ecPnt[13 573 483 833 446 195 039 386 191 823 126 470 - 200 557 258 027 456 445 688 282 344 784 336 - 103 649 212, 92 052 865 363 241 580 056 892 319 302 489 258 - 396 224 827 868 501 036 283 611 297 590 954 - 929 380 563], ecPnt[87 157 328 238 784 876 368 170 585 205 145 864 - 149 517 917 546 640 676 011 828 505 926 229 - 337 545 559, 49 329 002 166 834 480 429 318 163 327 266 459 - 724 785 883 670 773 200 913 837 835 518 032 - 372 564 965], ecPnt[71 520 343 767 673 936 517 024 801 860 388 366 - 635 511 719 271 546 226 054 794 719 093 802 - 183 956 346, 9 553 873 364 145 156 327 236 306 452 545 122 - 960 621 391 111 899 159 271 714 122 464 263 - 182 680 439], ecPnt[46 035 612 930 752 939 628 947 962 040 107 160 - 554 358 813 179 886 532 133 542 862 333 459 - 924 699 781, 45 447 377 390 389 063 397 803 533 683 113 774 - 914 322 866 417 884 852 592 242 983 262 329 - 077 405 730]}
$L_W := L_{2,W}.2$	Commitment to halves of $\overrightarrow{a_{2,W}}$ and $\overrightarrow{b_{2,W}}$ from P2	ecPnt[72 556 481 017 632 433 693 661 511 017 633 298 - 756 512 321 178 079 509 482 506 125 039 316 340 - 588 923, 58 628 123 834 064 766 575 081 733 647 402 276 - 204 937 487 939 674 442 001 533 559 083 399 118 - 278 801]
$R_W := R_{2,W}.2$	Commitment to other halves of $\overrightarrow{a_{2,W}}$ and $\overrightarrow{b_{2,W}}$ from P2	ecPnt[78 477 488 904 028 436 481 347 412 600 613 206 - 616 914 628 278 154 149 166 226 514 663 489 162 - 792 819, 6 247 703 021 684 575 120 229 172 459 504 646 297 - 671 321 951 456 904 911 106 626 166 428 084 591 - 726]

Generated random challenge

$u.2$ = 49778263016596358861024029034781411586030758570633068967576917901438334298455

Fold $\vec{G}$, $\vec{H}$, $\vec{B}$, $\vec{C}$, $\vec{D}$ and $\vec{P}$ based on challenge u_{IP}	
Name	**Value**
$u_{IP} := u_2.2$	49 778 263 016 596 358 861 024 029 034 781 411 586 - 030 758 570 633 068 967 576 917 901 438 334 298 455

$\overrightarrow{G_{next}} := u_{IP}^{-1}\overrightarrow{G_{lo}} + u_{IP}\overrightarrow{G_{hi}}$	{ecPnt[7 197 114 176 193 752 980 303 643 403 695 828 532 - 990 120 997 512 485 136 578 092 011 866 184 268 - 691, 60 730 773 379 641 461 649 895 581 236 702 273 487 - 306 200 766 726 745 877 010 587 530 816 841 971 - 454], ecPnt[25 740 309 652 130 294 792 014 274 623 822 906 233 - 778 698 385 632 087 797 179 590 756 586 611 447 - 326, 7 559 288 557 352 841 504 839 453 446 974 796 689 - 854 239 679 145 566 894 672 072 255 236 025 649 - 192]}
$\overrightarrow{H_{next}} := u_{IP}\overrightarrow{H_{lo}} + u_{IP}^{-1}\overrightarrow{H_{hi}}$	{ecPnt[9 923 669 315 589 724 236 175 802 580 426 592 379 - 857 281 435 169 619 817 581 664 784 806 305 050 - 600, 69 301 315 300 106 900 630 344 329 105 588 533 710 - 615 432 967 412 701 440 587 206 110 103 239 979 - 056], ecPnt[69 958 554 280 672 469 733 643 275 419 680 116 457 - 375 050 330 391 348 300 638 192 607 697 674 499 - 972, 70 322 526 074 710 939 661 538 785 781 005 409 111 - 808 935 480 016 459 677 331 324 303 468 832 670 - 948]}
$\overrightarrow{B_{next}} := u_{IP}\overrightarrow{B_{lo}} + u_{IP}^{-1}\overrightarrow{B_{hi}}$	{ecPnt[103 181 688 540 283 550 900 115 377 074 572 906 - 022 092 269 214 954 726 718 790 501 569 337 390 - 653 136, 88 587 317 905 811 582 887 852 333 128 737 362 967 - 001 668 115 722 517 496 378 551 323 743 472 873 - 003], ecPnt[78 043 950 849 939 684 905 494 886 280 107 655 203 - 568 705 298 939 663 195 960 427 464 870 794 481 - 036, 63 028 727 487 347 507 861 088 825 955 612 637 572 - 326 719 118 902 670 929 364 859 450 357 002 591 - 244]}
$\overrightarrow{C_{next}} := u_{IP}\overrightarrow{C_{lo}} + u_{IP}^{-1}\overrightarrow{C_{hi}}$	{ecPnt[77 648 398 834 955 981 434 915 225 171 372 670 033 - 920 110 798 382 972 995 557 496 536 072 160 616 - 152, 114 993 206 617 194 884 582 252 467 178 870 559 - 035 021 674 392 919 932 899 922 134 085 890 650 - 235 761], ecPnt[4 016 477 553 053 980 634 592 868 942 935 513 988 - 297 652 249 720 374 332 911 319 787 059 969 395 - 864, 107 780 355 073 054 246 547 274 550 757 710 943 - 490 511 352 511 246 624 914 557 077 087 278 354 - 014 724]}

$\overrightarrow{D_{next}} := u_{IP}\overrightarrow{D_{lo}} + u_{IP}^{-1}\overrightarrow{D_{hi}}$	{ecPnt[9 204 564 868 782 270 412 870 134 496 961 531 211 - 545 298 619 676 831 565 162 531 509 980 471 661 - 504, 110 656 730 936 786 158 948 279 040 537 731 806 - 719 194 876 348 740 960 648 492 508 893 706 752 - 655 534], ecPnt[57 318 399 460 579 651 382 855 744 160 054 965 144 - 349 045 280 754 541 796 339 058 133 751 450 351 - 507, 9 752 431 091 061 141 156 954 293 420 600 431 415 - 368 763 923 621 403 283 475 976 022 665 295 425 - 015]}
$P_{next,\{h,b,c,d,W\}} := P_{\{h,b,c,d,W\}} + u_{IP}^{2}L_{\{h,b,c,d,W\}} + u_{IP}^{-2}R_{\{h,b,c,d,W\}}$	{ecPnt[86 549 293 700 705 576 310 670 778 784 597 836 333 - 741 541 083 996 755 111 959 480 234 696 972 978 - 414, 20 933 742 971 864 474 163 704 352 952 168 549 433 - 873 232 361 691 192 046 047 306 574 101 250 855 - 678], ecPnt[5 329 237 281 080 061 557 087 386 541 512 094 989 - 216 765 361 219 651 754 727 494 075 651 069 853 - 882, 25 505 779 482 843 203 644 899 024 985 738 424 791 - 777 493 582 803 862 991 865 292 667 174 203 131 - 274], ecPnt[101 669 631 185 013 308 541 174 320 487 441 264 - 026 981 613 007 202 453 793 298 531 133 729 240 - 553 633, 82 960 685 798 015 610 060 110 994 730 438 408 801 - 163 464 994 126 881 961 038 986 643 726 868 054 - 985], ecPnt[80 930 121 240 166 102 505 541 582 335 909 769 099 - 418 550 745 921 470 989 327 339 028 870 473 478 - 673, 73 213 433 432 881 882 608 741 265 907 352 624 933 - 247 540 024 738 086 475 996 367 177 305 424 525 - 869], ecPnt[77 655 419 294 702 414 646 521 152 577 006 451 504 - 570 431 019 614 816 380 532 870 948 964 923 887 - 672, 68 497 091 790 293 927 373 067 739 936 022 331 983 - 241 210 497 850 650 557 960 541 940 114 992 397 - 821]}

Iteration 1 (Counting from 2 to 0)

Name	Semantics	Value
$k := k_2.1$	Current Iteration (counting down)	1
$n_2.1 = 2^k$	Current Length of Vectors	2

$\vec{G} := \overrightarrow{G_2}.1$		{ecPnt[7 197 114 176 193 752 980 303 643 403 695 828 532 - 990 120 997 512 485 136 578 092 011 866 184 268 - 691, 60 730 773 379 641 461 649 895 581 236 702 273 487 - 306 200 766 726 745 877 010 587 530 816 841 971 - 454], ecPnt[25 740 309 652 130 294 792 014 274 623 822 906 233 - 778 698 385 632 087 797 179 590 756 586 611 447 - 326, 7 559 288 557 352 841 504 839 453 446 974 796 689 - 854 239 679 145 566 894 672 072 255 236 025 649 - 192]}
$\vec{H} := \overrightarrow{H_2}.1$		{ecPnt[9 923 669 315 589 724 236 175 802 580 426 592 379 - 857 281 435 169 619 817 581 664 784 806 305 050 - 600, 69 301 315 300 106 900 630 344 329 105 588 533 710 - 615 432 967 412 701 440 587 206 110 103 239 979 - 056], ecPnt[69 958 554 280 672 469 733 643 275 419 680 116 457 - 375 050 330 391 348 300 638 192 607 697 674 499 - 972, 70 322 526 074 710 939 661 538 785 781 005 409 111 - 808 935 480 016 459 677 331 324 303 468 832 670 - 948]}
$\vec{B} := \overrightarrow{B_2}.1$		{ecPnt[103 181 688 540 283 550 900 115 377 074 572 906 022 - 092 269 214 954 726 718 790 501 569 337 390 653 - 136, 88 587 317 905 811 582 887 852 333 128 737 362 967 - 001 668 115 722 517 496 378 551 323 743 472 873 - 003], ecPnt[78 043 950 849 939 684 905 494 886 280 107 655 203 - 568 705 298 939 663 195 960 427 464 870 794 481 - 036, 63 028 727 487 347 507 861 088 825 955 612 637 572 - 326 719 118 902 670 929 364 859 450 357 002 591 - 244]}
$\vec{C} := \overrightarrow{C_2}.1$		{ecPnt[77 648 398 834 955 981 434 915 225 171 372 670 033 - 920 110 798 382 972 995 557 496 536 072 160 616 - 152, 114 993 206 617 194 884 582 252 467 178 870 559 035 - 021 674 392 919 932 899 922 134 085 890 650 235 - 761], ecPnt[4 016 477 553 053 980 634 592 868 942 935 513 988 - 297 652 249 720 374 332 911 319 787 059 969 395 - 864, 107 780 355 073 054 246 547 274 550 757 710 943 490 - 511 352 511 246 624 914 557 077 087 278 354 014 - 724]}

$\vec{D} := \overrightarrow{D_2}.1$		{ecPnt[9 204 564 868 782 270 412 870 134 496 961 531 211 - 545 298 619 676 831 565 162 531 509 980 471 661 - 504, 110 656 730 936 786 158 948 279 040 537 731 806 719 - 194 876 348 740 960 648 492 508 893 706 752 655 - 534], ecPnt[57 318 399 460 579 651 382 855 744 160 054 965 144 - 349 045 280 754 541 796 339 058 133 751 450 351 - 507, 9 752 431 091 061 141 156 954 293 420 600 431 415 - 368 763 923 621 403 283 475 976 022 665 295 425 - 015]}
$P_{\{h,b,c,d,W\}} := P_{2,\{h,b,c,d,W\}}.1$		{ecPnt[86 549 293 700 705 576 310 670 778 784 597 836 333 - 741 541 083 996 755 111 959 480 234 696 972 978 - 414, 20 933 742 971 864 474 163 704 352 952 168 549 433 - 873 232 361 691 192 046 047 306 574 101 250 855 - 678], ecPnt[5 329 237 281 080 061 557 087 386 541 512 094 989 - 216 765 361 219 651 754 727 494 075 651 069 853 - 882, 25 505 779 482 843 203 644 899 024 985 738 424 791 - 777 493 582 803 862 991 865 292 667 174 203 131 - 274], ecPnt[101 669 631 185 013 308 541 174 320 487 441 264 026 - 981 613 007 202 453 793 298 531 133 729 240 553 - 633, 82 960 685 798 015 610 060 110 994 730 438 408 801 - 163 464 994 126 881 961 038 986 643 726 868 054 - 985], ecPnt[80 930 121 240 166 102 505 541 582 335 909 769 099 - 418 550 745 921 470 989 327 339 028 870 473 478 - 673, 73 213 433 432 881 882 608 741 265 907 352 624 933 - 247 540 024 738 086 475 996 367 177 305 424 525 - 869], ecPnt[77 655 419 294 702 414 646 521 152 577 006 451 504 - 570 431 019 614 816 380 532 870 948 964 923 887 - 672, 68 497 091 790 293 927 373 067 739 936 022 331 983 - 241 210 497 850 650 557 960 541 940 114 992 397 - 821]}

Waiting to receive Lh.1 from Prover ...

Waiting to receive Rh.1 from Prover ...

Waiting to receive Lb.1 from Prover ...

Waiting to receive Rb.1 from Prover ...

Waiting to receive Lc.1 from Prover ...

Waiting to receive Rc.1 from Prover ...

Waiting to receive Ld.1 from Prover ...

Waiting to receive Rd.1 from Prover ...

Waiting to receive LW.1 from Prover ...

Waiting to receive RW.1 from Prover ...

Name	Semantics	Value
$L_{\{h,b,c,d\}} := L_{2,\{h,b,c,d\}}.1$	Commitments to halves of $\vec{a}$ and $\vec{b}$ based on $\{\vec{h},\vec{b},\vec{c},\vec{d}\}$ from P	{ecPnt[75 675 059 526 275 504 068 918 777 111 604 858 - 788 471 629 915 134 420 171 265 812 896 626 - 119 066 391, 18 197 204 939 082 113 188 285 923 802 867 554 - 462 472 120 668 263 296 596 875 421 190 375 - 977 857 454], ecPnt[12 416 377 022 545 922 744 939 982 468 289 996 - 520 934 072 056 023 744 431 017 501 656 184 - 389 730 319, 81 051 326 905 139 913 890 607 962 198 725 137 - 410 169 439 643 088 588 490 840 272 138 742 - 545 929 540], ecPnt[98 853 291 129 251 635 680 053 212 406 582 180 - 789 552 600 828 913 507 523 250 568 084 217 - 954 820 657, 87 281 355 932 318 972 300 122 755 145 147 993 - 995 828 597 934 957 482 135 707 810 979 826 - 096 889 712], ecPnt[89 851 019 805 865 117 834 787 478 702 313 840 - 467 099 246 847 828 862 988 794 868 493 385 - 234 963 271, 10 261 371 339 460 202 553 165 527 510 378 999 - 864 989 933 882 249 962 958 151 606 189 754 - 134 106 417]}

$R_{\{h,b,c,d\}} := R_{2,\{h,b,c,d\}}.1$	Commitments to other halves of $\vec{a}$ and $\vec{b}$ based on $\{\vec{h},\vec{b},\vec{c},\vec{d}\}$ from P	{ecPnt[30 549 668 247 877 315 168 586 354 189 454 040 - 046 905 352 468 501 612 133 078 991 262 937 - 458 834 121, 8 064 041 257 520 572 495 742 525 983 644 270 - 490 558 397 227 754 385 810 941 773 435 446 - 589 877 912], ecPnt[17 271 082 237 114 661 662 782 139 528 333 954 - 135 060 936 698 181 697 107 019 792 539 131 - 982 397 059, 100 485 113 882 867 413 113 226 576 224 258 467 - 945 632 855 040 297 438 669 404 557 630 187 - 333 726 261], ecPnt[77 039 539 825 354 912 829 621 507 432 128 099 - 080 227 116 487 899 322 574 405 235 729 421 - 193 756 314, 19 810 300 289 411 477 639 048 480 556 880 870 - 950 820 614 431 045 845 021 683 132 249 865 - 903 583 930], ecPnt[93 268 138 165 182 546 384 818 285 702 418 187 - 945 928 666 090 817 865 643 211 434 544 562 - 242 435 789, 111 279 597 227 286 293 277 376 134 144 323 743 - 374 311 220 963 861 626 278 010 175 736 815 - 569 060 927]}
$L_W := L_{2,W}.1$	Commitment to halves of $\overrightarrow{a_{2,W}}$ and $\overrightarrow{b_{2,W}}$ from P1	ecPnt[14 388 906 438 534 551 094 002 496 836 880 047 - 677 016 190 339 915 135 357 923 304 609 889 922 - 238 095, 25 350 077 021 118 276 994 302 587 444 563 735 - 933 315 591 300 912 219 383 404 434 462 160 307 - 924 273]
$R_W := R_{2,W}.1$	Commitment to other halves of $\overrightarrow{a_{2,W}}$ and $\overrightarrow{b_{2,W}}$ from P1	ecPnt[101 171 956 531 732 571 746 440 210 062 672 013 - 112 905 510 995 303 776 665 016 931 002 977 742 - 739 679, 95 465 264 805 104 010 998 282 450 451 339 127 - 840 467 517 182 363 902 674 875 441 902 949 420 - 100 229]

Generated random challenge

$u.1 = 60107283168128902360549399981388134586079393627328093194820115310250308340078$

Fold $\vec{G}$, $\vec{H}$, $\vec{B}$, $\vec{C}$, $\vec{D}$ and $\vec{P}$ based on challenge u_{IP}	
Name	**Value**
$u_{IP} := u_2.1$	60 107 283 168 128 902 360 549 399 981 388 134 586 - 079 393 627 328 093 194 820 115 310 250 308 340 078
$\overrightarrow{G_{next}} := u_{IP}^{-1}\overrightarrow{G_{lo}} + u_{IP}\overrightarrow{G_{hi}}$	{ecPnt[106 047 928 382 399 298 606 307 270 553 659 199 - 565 771 566 643 562 941 779 995 199 351 454 640 - 836 231, 93 720 323 795 988 478 677 394 413 863 093 059 165 - 226 732 196 254 855 469 832 899 039 748 827 794 - 813]}

$\overrightarrow{H_{next}} := u_{IP}\overrightarrow{H_{lo}} + u_{IP}^{-1}\overrightarrow{H_{hi}}$	{ecPnt[65 798 702 696 880 764 213 759 080 205 885 559 823 - 919 736 555 092 196 315 458 956 414 043 528 990 - 417, 113 060 727 953 871 578 706 455 950 897 122 407 - 255 435 350 435 014 258 540 357 290 902 460 111 - 718 764]}
$\overrightarrow{B_{next}} := u_{IP}\overrightarrow{B_{lo}} + u_{IP}^{-1}\overrightarrow{B_{hi}}$	{ecPnt[51 263 962 075 200 262 089 977 527 991 489 154 090 - 256 876 495 239 263 853 673 968 662 269 867 159 - 880, 59 304 354 354 125 370 257 888 899 770 035 709 731 - 793 990 653 314 774 277 689 815 845 108 336 596 - 654]}
$\overrightarrow{C_{next}} := u_{IP}\overrightarrow{C_{lo}} + u_{IP}^{-1}\overrightarrow{C_{hi}}$	{ecPnt[3 063 446 378 215 682 504 529 818 388 742 295 906 - 691 774 205 907 395 646 281 960 170 478 485 277 - 782, 41 711 340 372 428 252 715 592 184 575 370 511 221 - 524 887 064 978 818 572 894 341 476 773 483 215 - 453]}
$\overrightarrow{D_{next}} := u_{IP}\overrightarrow{D_{lo}} + u_{IP}^{-1}\overrightarrow{D_{hi}}$	{ecPnt[24 816 052 915 344 061 723 394 546 168 913 174 577 - 843 710 948 969 225 602 687 365 265 051 637 040 - 815, 67 401 455 371 277 781 651 260 916 671 100 331 294 - 882 258 933 658 861 019 459 735 037 385 044 630 - 090]}

$P_{next,\{h,b,c,d,W\}} := P_{\{h,b,c,d,W\}} + u_{IP}^{2} L_{\{h,b,c,d,W\}} + u_{IP}^{-2} R_{\{h,b,c,d,W\}}$	{ecPnt[89 558 932 688 492 177 722 272 018 113 494 238 314 - 242 283 658 549 690 867 267 807 775 428 511 718 - 321, 82 308 534 977 355 512 322 370 152 675 903 609 108 - 741 121 212 616 595 116 137 887 645 130 578 579 - 276], ecPnt[2 535 106 563 306 329 634 217 460 360 590 660 394 - 532 660 318 333 598 601 313 805 246 253 811 670 - 167, 37 905 362 638 224 769 609 916 560 569 773 911 176 - 947 719 296 274 603 389 536 502 708 617 675 361 - 051], ecPnt[112 719 119 487 721 161 889 159 211 082 551 290 - 789 635 633 197 790 401 234 517 080 402 764 506 - 322 042, 59 038 507 861 533 321 775 922 462 484 860 668 668 - 411 724 675 754 721 170 660 930 498 202 940 497 - 565], ecPnt[105 479 895 284 880 952 363 906 946 754 971 019 - 937 646 107 950 192 208 454 804 756 816 712 996 - 878 277, 88 367 073 377 840 609 503 965 989 028 276 026 025 - 955 631 419 100 797 652 975 414 156 539 639 740 - 891], ecPnt[66 576 517 566 997 309 876 776 372 876 836 199 019 - 185 475 935 761 406 744 525 505 228 996 757 486 - 023, 90 324 928 969 768 165 511 842 292 293 833 554 044 - 922 227 263 120 569 797 649 525 696 262 179 099 - 046]}

Waiting to receive a0 from Prover ...

Waiting to receive b0 from Prover ...

Waiting to receive a0W from Prover ...

Waiting to receive b0W from Prover ...

Iteration 0 (Counting from 2 to 0)			
Name	Semantics	**Value**	
$n_0 := n_2.0[1] \stackrel{?}{=} 1$	Halving Complete?	True	
$k_0 := k_2.0[1] \stackrel{?}{=} 0$	Halving Complete?	P,V	True
$G_0 := \overrightarrow{G_2}.0[1]$		ecPnt[106 047 928 382 399 298 606 307 - 270 553 659 199 565 771 566 - 643 562 941 779 995 199 351 - 454 640 836 231, 93 720 323 795 988 478 677 394 - 413 863 093 059 165 226 732 - 196 254 855 469 832 899 039 - 748 827 794 813]	

$H_0 := \overrightarrow{H_2}.0[1]$		ecPnt[65 798 702 696 880 764 213 759 - 080 205 885 559 823 919 736 - 555 092 196 315 458 956 414 - 043 528 990 417, 113 060 727 953 871 578 706 455 - 950 897 122 407 255 435 350 - 435 014 258 540 357 290 902 - 460 111 718 764]	
$B_0 := \overrightarrow{B_2}.0[1]$		ecPnt[51 263 962 075 200 262 089 977 - 527 991 489 154 090 256 876 - 495 239 263 853 673 968 662 - 269 867 159 880, 59 304 354 354 125 370 257 888 - 899 770 035 709 731 793 990 - 653 314 774 277 689 815 845 - 108 336 596 654]	
$C_0 := \overrightarrow{C_2}.0[1]$		ecPnt[3 063 446 378 215 682 504 529 - 818 388 742 295 906 691 774 - 205 907 395 646 281 960 170 - 478 485 277 782, 41 711 340 372 428 252 715 592 - 184 575 370 511 221 524 887 - 064 978 818 572 894 341 476 - 773 483 215 453]	
$D_0 := \overrightarrow{D_2}.0[1]$		ecPnt[24 816 052 915 344 061 723 394 - 546 168 913 174 577 843 710 - 948 969 225 602 687 365 265 - 051 637 040 815, 67 401 455 371 277 781 651 260 - 916 671 100 331 294 882 258 - 933 658 861 019 459 735 037 - 385 044 630 090]	
a_0		108 423 123 045 658 025 157 618 - 639 588 976 361 816 248 958 028 - 757 952 488 070 957 825 485 293 - 867 099	
b_0		42 898 793 650 488 454 998 791 - 851 268 901 023 693 781 135 981 - 625 855 329 285 685 978 384 189 - 744 313	
$a_{0,W}$		1 375 560 257 280 688 522 612 - 389 858 362 182 280 930 062 576 - 340 977 657 137 703 999 687 345 - 468 197	
$b_{0,W}$		77 752 142 049 699 130 174 929 - 194 338 535 297 301 448 986 784 - 213 237 165 185 139 643 741 664 - 012 763	

$P_{0,\{h,b,c,d,W\}} := P_{2,\{h,b,c,d,W\}}.0$		{ecPnt[89 558 932 688 492 177 722 - 272 018 113 494 238 314 242 - 283 658 549 690 867 267 807 - 775 428 511 718 321, 82 308 534 977 355 512 322 - 370 152 675 903 609 108 741 - 121 212 616 595 116 137 887 - 645 130 578 579 276], ecPnt[2 535 106 563 306 329 634 217 - 460 360 590 660 394 532 660 - 318 333 598 601 313 805 246 - 253 811 670 167, 37 905 362 638 224 769 609 - 916 560 569 773 911 176 947 - 719 296 274 603 389 536 502 - 708 617 675 361 051], ecPnt[112 719 119 487 721 161 889 - 159 211 082 551 290 789 635 - 633 197 790 401 234 517 080 - 402 764 506 322 042, 59 038 507 861 533 321 775 - 922 462 484 860 668 668 411 - 724 675 754 721 170 660 930 - 498 202 940 497 565], ecPnt[105 479 895 284 880 952 363 - 906 946 754 971 019 937 646 - 107 950 192 208 454 804 756 - 816 712 996 878 277, 88 367 073 377 840 609 503 - 965 989 028 276 026 025 955 - 631 419 100 797 652 975 414 - 156 539 639 740 891], ecPnt[66 576 517 566 997 309 876 - 776 372 876 836 199 019 185 - 475 935 761 406 744 525 505 - 228 996 757 486 023, 90 324 928 969 768 165 511 - 842 292 293 833 554 044 922 - 227 263 120 569 797 649 525 - 696 262 179 099 046]}	
$c_0 := a_0 b_0$		50 802 272 133 873 189 188 778 - 463 155 973 251 716 683 768 205 - 680 082 583 600 857 092 767 270 - 065 867	
$c_{0,W} := a_{0,W} b_{0,W}$		70 417 494 905 687 723 108 590 - 833 688 312 608 761 422 205 629 - 541 919 203 657 017 687 229 748 - 318 131	

$P_{0,\{h,b,c,d,W\}} \stackrel{?}{=} P_{\{h,b,c,d,W\}} + \langle u_2^2, L_{2,\{h,b,c,d,W\}}\rangle + \langle u_2^{-2}, R_{2,\{h,b,c,d,W\}}\rangle$	Sanity Check for P_0	{True, True, True, True}	
$P_{0,\{h,b,c,d\}} \stackrel{?}{=} G_0 a_0 + \{H,B,C,D\}_0 b_0 + c_0 g'$	**Check No.2{h,b,c,d} for Verification of Shuffle Proof (with Inner Product Protocol) ok?**	{True, True, True, True}	
$P_{0,W} \stackrel{?}{=} G_0 a_{0,W} + H_0 b_{0,W} + c_{0,W} g'$	**Check No.2W for Verification of PR Proof (with Inner Product Protocol) ok?**	True	

Inner product protocol finished.

Total Communication Cost			
Type	**Count**	**Names**	**Bits**
group	34	V, W, AW, SW, AL, SL, SRh, SRb, SRc, SRd, T3, T5, T1W, T2W, Lh.2, Rh.2, Lb.2, Rb.2, Lc.2, Rc.2, Ld.2, Rd.2, LW.2, RW.2, Lh.1, Rh.1, Lb.1, Rb.1, Lc.1, Rc.1, Ld.1, Rd.1, LW.1, RW.1	8738
scalar	21	r, u, y, z, w, x, taux, muh, mub, muc, mud, t, tauxW, muW, tW, u.2, u.1, a0, b0, a0W, b0W	5376
total	55		14 114

Verification succeeded.

Inspection of Pedersen Commitments (Multi-Key Shuffle with Inner Product Protocol)

Request inspection of Pedersen commitments V, A, S, T_1 and T_2 (range protocol) as well as $L[n \ldots 1]$ and $R[n \ldots 1]$ (inner product protocol) from prover to verify that his commitments were done correctly. Additionally (but not necessary), check other variables exchanged during protocol execution.

In[323]:=

```
InnerProductArgumentInspection[inspectionReply_] := Module[
    {success,
    LhInspectionFailures,RhInspectionFailures,
    LbInspectionFailures,RbInspectionFailures,
    LcInspectionFailures,RcInspectionFailures,
    LdInspectionFailures,RdInspectionFailures,
    LWInspectionFailures,RWInspectionFailures,
    a0WDefinitionOk,b0WDefinitionOk,
    a0DefinitionOk,b0DefinitionOk},
    (*Check commitments L and R of every iteration of the inner product protocol
    and the fully halved version of a⃗ and b⃗, a₀ and b₀.*)

    a⃗IP = Join[i⃗[[1;;Min[{n_pow2,n}]]],ConstantArray[0,Max[0,n_pow2-n]]];
    b⃗IP = Join[r⃗[[1;;Min[{n_pow2,n}]]],ConstantArray[0,Max[0,n_pow2-n]]];
```

```
\vec{a}_W = Join[\vec{l}_W[[1;;Min[{n_pow2,n}]]],ConstantArray[0,Max[0,n_pow2-n]]];
\vec{b}_W = Join[\vec{r}_W[[1;;Min[{n_pow2,n}]]],ConstantArray[0,Max[0,n_pow2-n]]];

currentIteration = 1;
LhInspectionFailures = {};
RhInspectionFailures = {};
LbInspectionFailures = {};
RbInspectionFailures = {};
LcInspectionFailures = {};
RcInspectionFailures = {};
LdInspectionFailures = {};
RdInspectionFailures = {};
LWInspectionFailures = {};
RWInspectionFailures = {};

While[n_2[[currentIteration]] ≠ 1,
    ciStr = IntegerString[k_2[[currentIteration]]];
    currentN = n_2[[currentIteration]];
    halfN = currentN/2;
    k = k_2[[currentIteration]];
    \vec{G} = \vec{G_2}[[currentIteration]];
    \vec{H} = \vec{H_2}[[currentIteration]];
    \vec{B} = \vec{B_2}[[currentIteration]];
    \vec{C} = \vec{C_2}[[currentIteration]];
    \vec{D} = \vec{D_2}[[currentIteration]];
    P_h = P_2,h[[currentIteration]];
    P_b = P_2,b[[currentIteration]];
    P_c = P_2,c[[currentIteration]];
    P_d = P_2,d[[currentIteration]];
    P_W = P_2,W[[currentIteration]];
    (*Split into lower and higher parts*)
    \vec{G_lo} = \vec{G}[[1;;halfN]];
    \vec{G_hi} = \vec{G}[[halfN+1;;-1]];
    \vec{H_lo} = \vec{H}[[1;;halfN]];
    \vec{H_hi} = \vec{H}[[halfN+1;;-1]];
    \vec{B_lo} = \vec{B}[[1;;halfN]];
    \vec{B_hi} = \vec{B}[[halfN+1;;-1]];
    \vec{C_lo} = \vec{C}[[1;;halfN]];
    \vec{C_hi} = \vec{C}[[halfN+1;;-1]];
    \vec{D_lo} = \vec{D}[[1;;halfN]];
    \vec{D_hi} = \vec{D}[[halfN+1;;-1]];
    \vec{a_lo} = \vec{a_IP}[[1;;halfN]];
    \vec{a_hi} = \vec{a_IP}[[halfN+1;;-1]];
    \vec{b_lo} = \vec{b_IP}[[1;;halfN]];
    \vec{b_hi} = \vec{b_IP}[[halfN+1;;-1]];
    \vec{a_lo,W} = \vec{a_W}[[1;;halfN]];
    \vec{a_hi,W} = \vec{a_W}[[halfN+1;;-1]];
    \vec{b_lo,W} = \vec{b_W}[[1;;halfN]];
    \vec{b_hi,W} = \vec{b_W}[[halfN+1;;-1]];
```

```
    (*Digest into L and R*)
    L_h = a_lo . G_hi + b_hi . H_lo + a_lo . b_hi * g';
    R_h = a_hi . G_lo + b_lo . H_hi + a_hi . b_lo * g';
    L_b = a_lo . G_hi + b_hi . B_lo + a_lo . b_hi * g';
    R_b = a_hi . G_lo + b_lo . B_hi + a_hi . b_lo * g';
    L_c = a_lo . G_hi + b_hi . C_lo + a_lo . b_hi * g';
    R_c = a_hi . G_lo + b_lo . C_hi + a_hi . b_lo * g';
    L_d = a_lo . G_hi + b_hi . D_lo + a_lo . b_hi * g';
    R_d = a_hi . G_lo + b_lo . D_hi + a_hi . b_lo * g';
    L_W = a_lo,W . G_hi,W + b_hi,W . C_lo,W + a_lo,W . b_hi,W * g';
    R_W = a_hi,W . G_lo,W + b_lo,W . C_hi,W + a_hi,W . b_lo,W * g';
    (*Check L and R*)
    If[L_h ≠ L_2,h[[currentIteration]], AppendTo[LhInspectionFailures, Lh<>"."<>ciStr]];
    If[R_h ≠ R_2,h[[currentIteration]], AppendTo[RhInspectionFailures, Rh<>"."<>ciStr]];
    If[L_b ≠ L_2,b[[currentIteration]], AppendTo[LbInspectionFailures, Lb<>"."<>ciStr]];
    If[R_b ≠ R_2,b[[currentIteration]], AppendTo[RbInspectionFailures, Rb<>"."<>ciStr]];
    If[L_c ≠ L_2,c[[currentIteration]], AppendTo[LcInspectionFailures, Lc<>"."<>ciStr]];
    If[R_c ≠ R_2,c[[currentIteration]], AppendTo[RcInspectionFailures, Rc<>"."<>ciStr]];
    If[L_d ≠ L_2,d[[currentIteration]], AppendTo[LdInspectionFailures, Ld<>"."<>ciStr]];
    If[R_d ≠ R_2,d[[currentIteration]], AppendTo[RdInspectionFailures, Rd<>"."<>ciStr]];
    If[L_W ≠ L_2,W[[currentIteration]], AppendTo[LWInspectionFailures, LW<>"."<>ciStr]];
    If[R_W ≠ R_2,W[[currentIteration]], AppendTo[RWInspectionFailures, RW<>"."<>ciStr]];
    u_IP = u_2[[currentIteration]];
    u^-1 = modinvq[u_IP];
    a_IP = modq[u_IP * a_lo + u^-1 * a_hi];
    b_IP = modq[u^-1 * b_lo + u_IP * b_hi];
    a_W = modq[u_IP * a_lo,W + u^-1 * a_hi,W];
    b_W = modq[u^-1 * b_lo,W + u_IP * b_hi,W];
    currentIteration += 1;
];
(*Last, single-valued iteration*)
a0DefinitionOk = (a_0 == a_IP[[1]]);
b0DefinitionOk = (b_0 == b_IP[[1]]);
a0WDefinitionOk = (a_0,W == a_W[[1]]);
b0WDefinitionOk = (b_0,W == b_W[[1]]);
Print[Grid[{
    {Style["Expression",Bold], Style["Type of Check",Bold], Style["Value",Bold]},
    {
        "L_{h,b,c,d,W},i ≟ <a_i_lo{,W}, G_i_hi> + <b_i_hi{,W}, {H,B,C,D,H}_i_lo> + <a_i_lo{,W}, b_i_hi{,W}>g'\ni∈[⌈log_2(n)
        "Commitment",
        {
            If[Length[LhInspectionFailures]==0,True,"Failures: "<>StringRiffle[LhInspectio
            If[Length[LbInspectionFailures]==0,True,"Failures: "<>StringRiffle[LbInspectio
            If[Length[LcInspectionFailures]==0,True,"Failures: "<>StringRiffle[LcInspectio
            If[Length[LdInspectionFailures]==0,True,"Failures: "<>StringRiffle[LdInspectio
            If[Length[LWInspectionFailures]==0,True,"Failures: "<>StringRiffle[LWInspectio
        }
```

```
        },
        {
            "R_{h,b,d,c,W},i ≟ <a_{hi,W}, G_{lo}> + <b_{lo,W}, {H,B,D,C,H}_{hi}> + <a_{hi,W}, b_{lo,W}>g'\nie[⌈log_2(n)
            "Commitment",
            {
                If[Length[RhInspectionFailures]==0,True,"Failures: "<>StringRiffle[RhInspectio
                If[Length[RbInspectionFailures]==0,True,"Failures: "<>StringRiffle[RbInspectio
                If[Length[RcInspectionFailures]==0,True,"Failures: "<>StringRiffle[RcInspectio
                If[Length[RdInspectionFailures]==0,True,"Failures: "<>StringRiffle[RdInspectio
                If[Length[RWInspectionFailures]==0,True,"Failures: "<>StringRiffle[RWInspectio
            }
        },
        {"a_0 ≟ successively halved a_{⌈log_2(n)⌉}", "Definition", a0DefinitionOk},
        {"b_0 ≟ successively halved b_{⌈log_2(n)⌉}", "Definition", b0DefinitionOk},
        {"a_{0,W} ≟ successively halved a_{⌈log_2(n)⌉,W}", "Definition", a0WDefinitionOk},
        {"b_{0,W} ≟ successively halved b_{⌈log_2(n)⌉,W}", "Definition", b0WDefinitionOk}
    }, Alignment→{{Left,Left,Left}},Frame→All]];

    success = False;
    Which[
        Length[LhInspectionFailures] > 0, InspectionFailed[StringRiffle[LhInspectionFailures,
        Length[RhInspectionFailures] > 0, InspectionFailed[StringRiffle[RhInspectionFailures,
        Length[LbInspectionFailures] > 0, InspectionFailed[StringRiffle[LbInspectionFailures,
        Length[RbInspectionFailures] > 0, InspectionFailed[StringRiffle[RbInspectionFailures,
        Length[LcInspectionFailures] > 0, InspectionFailed[StringRiffle[LcInspectionFailures,
        Length[RcInspectionFailures] > 0, InspectionFailed[StringRiffle[RcInspectionFailures,
        Length[LdInspectionFailures] > 0, InspectionFailed[StringRiffle[LdInspectionFailures,
        Length[RdInspectionFailures] > 0, InspectionFailed[StringRiffle[RdInspectionFailures,
        Length[LWInspectionFailures] > 0, InspectionFailed[StringRiffle[LWInspectionFailures,
        Length[RWInspectionFailures] > 0, InspectionFailed[StringRiffle[RWInspectionFailures,
        !a0DefinitionOk, InspectionFailed["Definition of a0"],
        !b0DefinitionOk, InspectionFailed["Definition of b0"],
        !a0WDefinitionOk, InspectionFailed["Definition of a0W"],
        !b0WDefinitionOk, InspectionFailed["Definition of b0W"],
        True, success = True
    ];
    success
]

If[fiatShamirSingleKernel,
    Print["Can not send inspection requests in Single-Kernel mode. Done."];
(*Else*),
    TellProver[{"_inspection_request_"→True}];
    inspectionReply = WaitForVariable["_inspection_reply","Prover"];
    (*Implementing timeout on wait not possible in Mathematica but needed in practice*)
    If[MultiKeyShuffleArgumentInspection[inspectionReply],
        If[InnerProductArgumentInspection[inspectionReply],
            InspectionSucceeded[];
        ];
    ];
```

Waiting to receive _inspection_reply from Prover ...

Expression	Type of Check	Value
π	Permutation	{3, 1, 2}
s	Shift	77 055 416 219 496 477 297 458 718 029 586 723 - 439 661 366 871 894 090 478 002 190 167 067 968 - 031 603
$A_W \stackrel{?}{=} \langle \overrightarrow{a_{L,W}}, \vec{g} \rangle + \langle \overrightarrow{a_{R,W}}, \vec{h} \rangle + \alpha_W h$	Inspection	True
$S_W \stackrel{?}{=} \langle \overrightarrow{s_{L,W}}, \vec{g} \rangle + \langle \overrightarrow{s_{R,W}}, \vec{h} \rangle + \beta_W h$	Inspection	True
$\overrightarrow{h'} \stackrel{?}{=} s_\pi(\vec{h})$	Shifted permutation	True
$\overrightarrow{b'} \stackrel{?}{=} s_\pi(\vec{b})$	Shifted permutation	True
$\overrightarrow{c'} \stackrel{?}{=} s_\pi(\vec{c})$	Shifted permutation	True
$\overrightarrow{d'} \stackrel{?}{=} s_\pi(\vec{d})$	Shifted permutation	True
$V \stackrel{?}{=} sg$	Inspection	True
$W \stackrel{?}{=} s^n g + \sigma_W h$	Inspection	True
$\overrightarrow{a_R} :\equiv_q s_{\pi^{-1}}(\vec{k})$ $\overrightarrow{a_L} :\equiv_q$ Multiplicative aggregation of $\overrightarrow{a_L}$ starting at 1		
α		17 398 176 202 481 637 751 860 114 248 176 047 - 760 531 156 697 850 321 616 127 992 279 049 328 - 252 413
β		27 219 552 920 418 278 154 622 008 378 385 320 - 785 396 869 030 159 780 024 013 163 024 974 396 - 033 246
ρ_h		102 986 942 228 350 997 637 870 447 182 702 336 - 728 859 860 977 331 651 452 906 247 885 964 411 - 195 770
ρ_b		67 750 394 590 451 778 158 204 940 167 618 240 - 646 217 008 969 775 154 608 541 150 812 022 072 - 315 973
ρ_c		74 713 749 057 188 690 237 921 443 254 967 297 - 886 915 577 695 515 183 681 394 976 026 396 939 - 778 234
ρ_d		55 739 259 823 876 949 789 125 654 739 186 763 - 620 164 839 911 902 539 974 830 065 163 733 528 - 027 333
$\overrightarrow{s_L}$		{3 490 043 951 473 342 957 863 269 612 888 825 - 370 966 098 959 732 202 668 461 366 396 201 - 773 608 447, 81 026 870 043 438 310 759 789 738 090 565 752 - 971 348 726 422 705 469 591 555 083 035 827 - 778 970 747, 109 413 017 180 886 544 487 921 683 678 413 534 - 566 795 499 839 518 913 016 004 076 708 382 - 233 891 835}

$\vec{s_R}$		{12 507 601 888 368 634 928 041 736 808 448 324 - 961 591 369 924 503 374 677 351 026 235 158 - 490 466 997, 63 313 909 303 783 745 204 827 875 316 592 779 - 921 398 731 751 274 269 282 817 433 014 526 - 141 649 354, 27 472 217 458 736 686 113 325 566 075 209 874 - 854 922 345 264 255 234 592 930 381 510 746 - 121 210 239}
$A_L \stackrel{?}{=} <\vec{a_L},\vec{g}> + \alpha h$	Inspection	True
$S_L \stackrel{?}{=} <\vec{s_L},\vec{g}> + \beta h$	Inspection	True
$S_{R,h} \stackrel{?}{=} <\vec{s_R},\vec{h}> + \rho_h h$	Inspection	True
$S_{R,b} \stackrel{?}{=} <\vec{s_R},\vec{b}> + \rho_b h$	Inspection	True
$S_{R,c} \stackrel{?}{=} <\vec{s_R},\vec{c}> + \rho_c h$	Inspection	True
$S_{R,d} \stackrel{?}{=} <\vec{s_R},\vec{d}> + \rho_d h$	Inspection	True
$\vec{l_0} :\equiv_q \vec{a_L} + z^3\vec{y^{-n}}$ $\vec{l_2} :\equiv_q \vec{s_L}$ $\vec{r_1} :\equiv_q \vec{y^n}\circ\vec{a_R} - \vec{y_z^{n-1}}$ $\vec{r_3} :\equiv_q \vec{y^n}\circ\vec{s_R}$		
t_3		17 310 733 161 837 013 592 816 310 149 157 017 - 810 465 542 261 843 218 431 452 323 969 115 777 - 752 129
τ_3		103 025 787 400 053 846 491 173 493 263 859 547 - 421 436 948 279 868 469 816 606 692 097 792 288 - 570 268
$T_3 \stackrel{?}{=} gt_3 + h\tau_3$	Inspection	True
$t_3 \stackrel{?}{\equiv}_q <\vec{l_0},\vec{r_3}> + <\vec{l_2},\vec{r_1}>$	Definition	True
t_5		79 350 839 946 051 578 713 775 231 271 034 862 - 562 040 122 148 254 790 044 338 660 946 167 057 - 333 318
τ_5		95 538 448 397 885 489 565 925 196 202 909 322 - 984 569 275 766 270 781 222 063 297 280 902 331 - 526 819
$T_5 \stackrel{?}{=} gt_5 + h\tau_5$	Inspection	True
$t_5 \stackrel{?}{\equiv}_q <\vec{l_2},\vec{r_3}>$	Definition	True
$\vec{l_{0,W}} :\equiv_q \vec{a_{L,W}} + z\vec{y^{-n}} + \vec{w^n} - \vec{w_0^{n-1}}y^{-1}$ $\vec{l_{1,W}} :\equiv_q \vec{s_{L,W}}$ $\vec{r_{0,W}} :\equiv_q \vec{y^n}\circ\vec{a_{R,W}} - \vec{y_z^{n-1}}$ $\vec{r_{1,W}} :\equiv_q \vec{y^n}\circ\vec{s_{R,W}}$		
$t_{1,W}$		42 934 228 482 704 277 171 589 913 215 541 505 - 806 599 900 158 099 704 059 586 670 717 978 694 - 400 618
$\tau_{1,W}$		91 690 863 698 252 499 759 127 603 862 000 154 - 779 781 812 325 324 670 430 965 064 128 712 168 - 545 446
$T_{1,W} \stackrel{?}{=} gt_{1,W} + h\tau_{1,W}$	Inspection	True

$t_{1,W} \stackrel{?}{\equiv}_q \langle\overrightarrow{l_{0,W}}, \overrightarrow{r_{1,W}}\rangle + \langle\overrightarrow{l_{1,W}}, \overrightarrow{r_{0,W}}\rangle$	Definition	True
$t_{2,W}$		31 797 358 138 604 843 034 824 747 433 975 683 - 328 471 314 573 082 373 928 816 157 681 987 041 - 467 600
$\tau_{2,W}$		4 010 814 831 465 664 235 548 265 476 115 974 139 - 001 360 816 736 986 103 882 851 010 375 142 831 - 463
$T_{2,W} \stackrel{?}{=} gt_{2,W} + h\tau_{2,W}$	Inspection	True
$t_{2,W} \stackrel{?}{\equiv}_q \langle\overrightarrow{l_{1,W}}, \overrightarrow{r_{1,W}}\rangle$	Definition	True
$\vec{l} :\equiv_q \overrightarrow{l_0} + \overrightarrow{l_2}x^2$ $\vec{r} :\equiv_q \overrightarrow{r_1}x + \overrightarrow{r_3}x^3$		
$\overrightarrow{l_W} :\equiv_q \overrightarrow{l_{0,W}} + \overrightarrow{l_{1,W}}x$ $\overrightarrow{r_W} :\equiv_q \overrightarrow{r_{0,W}} + \overrightarrow{r_{1,W}}x$		
$\mu_h \stackrel{?}{\equiv}_q \alpha + \beta x^2 + \rho_h x^3$	Definition	True
$\mu_b \stackrel{?}{\equiv}_q \alpha + \beta x^2 + \rho_b x^3$	Definition	True
$\mu_c \stackrel{?}{\equiv}_q \alpha + \beta x^2 + \rho_c x^3$	Definition	True
$\mu_d \stackrel{?}{\equiv}_q \alpha + \beta x^2 + \rho_d x^3$	Definition	True
$\tau_x \stackrel{?}{\equiv}_q \sigma_W y^{n-1}\hat{k}x + \tau_3 x^3 + \tau_5 x^5$	Definition	True
$\hat{t} \stackrel{?}{\equiv}_q \langle\vec{l}, \vec{r}\rangle$	Definition	True
$\tau_{x,W} \stackrel{?}{\equiv}_q \sigma_W y^{n-1} + \tau_{1,W}x + \tau_{2,W}x^2$	Definition	True
$\hat{t_W} \stackrel{?}{\equiv}_q \langle\overrightarrow{l_W}, \overrightarrow{r_W}\rangle$	Definition	True

Expression	**Type of Check**	**Value**
$L_{\{h,b,c,d,W\},i} \stackrel{?}{=} \langle\overrightarrow{a_{i_{lo\{,W\}}}}, \overrightarrow{G_{i_{hi}}}\rangle + \langle\overrightarrow{b_{i_{hi\{,W\}}}}, \overrightarrow{\{H, B, C, D, H\}_{i_{lo}}}\rangle + \langle\overrightarrow{a_{i_{lo\{,W\}}}}, \overrightarrow{b_{i_{hi\{,W\}}}}\rangle g'$ $i \in [\lceil \log_2(n) \rceil ... 1]$	Commitment	{True, True, True, True, True}
$R_{\{h,b,d,c,W\},i} \stackrel{?}{=} \langle\overrightarrow{a_{i_{hi\{,W\}}}}, \overrightarrow{G_{i_{lo}}}\rangle + \langle\overrightarrow{b_{i_{lo\{,W\}}}}, \overrightarrow{\{H, B, D, C, H\}_{i_{hi}}}\rangle + \langle\overrightarrow{a_{i_{hi\{,W\}}}}, \overrightarrow{b_{i_{lo\{,W\}}}}\rangle g'$ $i \in [\lceil \log_2(n) \rceil ... 1]$	Commitment	{True, True, True, True, True}
$a_0 \stackrel{?}{=}$ successively halved $\overrightarrow{a_{\lceil \log_2(n) \rceil}}$	Definition	True
$b_0 \stackrel{?}{=}$ successively halved $\overrightarrow{b_{\lceil \log_2(n) \rceil}}$	Definition	True
$a_{0,W} \stackrel{?}{=}$ successively halved $\overrightarrow{a_{\lceil \log_2(n) \rceil, W}}$	Definition	True
$b_{0,W} \stackrel{?}{=}$ successively halved $\overrightarrow{b_{\lceil \log_2(n) \rceil, W}}$	Definition	True

Inspection succeeded.