# The protocols of Sharemind

# Sharemind system

- Three computing parties (called “miners”). One may be corrupt.
- Semi-honest adversary.
- Secure channels between each pair of parties.
- Unconditionally* secure.
  - Security of channels?
  - Source of randomness?
- Data providers share their inputs for the miners.
- Controller traverses the circuit of $f$ and instructs the miners.

# Sharing

- The values are from a finite ring $R$.
  - In Sharemind platform, $R = \mathbb{Z}_{2^{32}}$.
- The arithmetic circuit for $f$ is made up of operations of $R$.
- The values are shared additively:
  - $v \in R$ is shared as $(s_0, s_1, s_2) \in R^3$, where $s_0 + s_1 + s_2 = v$, but any two shares look like uniformly distributed independent random values.
  - $i$-th miner knows $s_i$.
- A data provider shares $v$ by
  - randomly generating $s_0, s_1 \in_R R$;
  - defining $s_2 = v - s_0 - s_1$;
  - sending $s_i$ to miner $M_i$.
- Note that none of the actions of a data provider qualifies as cheating.

# Resharing a value

- Let $v$ be shared as $s_0 + s_1 + s_2$.
- We want to have a different sharing $v = t_0 + t_1 + t_2$, such that $t_i$ is independent of $s_i$.
- Protocol:
    - $P_i$ generates $r_i \in_R R$ and sends it to $P_{(i+1) \bmod 3}$;
        - $P_i$ receives $r_{(i-1) \bmod 3}$
    - $P_i$ sets $t_i = s_i + r_i - r_{(i-1) \bmod 3}$.
- An important sub-protocol: makes a share of a value independent of other shares and uniformly distributed.

# Non-interactive protocols

- To add two shared values or to multiply with a scalar: each miner does the same operation with the shares it holds.
- To open a shared value: each miner sends its share to the controller.

# Ideal functionality $\mathcal{I}$

- Reactive — several rounds between $\mathcal{I}$ and the environment.
- Keeps of database of values $D : \mathbb{N} \to R \cup \{\perp\}$.
  - Elements of $\mathbb{N}$ — handles.
  - Let $\ell_D$ be the index of the last filled slot of $D$, initially $0$.
- Environment $H$ gives commands to $\mathcal{I}$, receives answers:
- Command store$(v)$, $v \in R$:
  - $D[$++$\ell_D] := v$; return $\ell_D$.
- Command retrieve$(h)$:
  - return $D[h]$.
- Command $\star(h_1, \ldots, h_k)$, where $\star$ is $k$-ary arithmetic operator:
  - $D[$++$\ell_D] = \star(D[h_1], \ldots, D[h_k])$; return $\ell_D$.
- $\mathcal{I}$ sends all executed commands to the adversary $\mathcal{A}_{\text{ideal}}$.
- $H$ and $\mathcal{A}_{\text{ideal}}$ can talk to each other directly.

# Real functionality

- Environment $H$ talks to the controller $\mathcal{C}$. Controller talks with the miners.
  - $\mathcal{C}$ basically forwards the commands to miners.
- Controller forwards all executed commands to the adversary $\mathcal{A}_{\mathsf{real}}$.
- If some $M_i$ is corrupted then continuously sends all of its internal state to $\mathcal{A}_{\mathsf{real}}$.
- Each miner $M_i$ keeps a database $D_i : \mathbb{N} \to R \cup \{\perp\}$.
  - The database stores the shares of the values.
- $H$ and $\mathcal{A}_{\mathsf{real}}$ can talk to each other directly.

# Security

Black-box reactive simulatability:

- There must exist a simulator $Sim$, such that
- For any $H$ and $\mathcal{A}_{\mathsf{real}}$
- If we define $\mathcal{A}_{\mathsf{ideal}} = Sim \mid \mathcal{A}_{\mathsf{real}}$ then
- $H$ cannot distinguish whether it is running in parallel with
  - $\mathcal{C}$, $M_0, M_1, M_2$, $\mathcal{A}_{\mathsf{real}}$; or
  - $\mathcal{I}$, $\mathcal{A}_{\mathsf{ideal}}$.

Important: $Sim$ must work during the runtime of the protocol, not afterwards.

# Simulating simple commands

- Let $M_c$ be corrupt, $c \in \{0,1,2\}$.
- Receiving store$(v)$ from $\mathcal{I}$:
  - Forward store$(v)$ to $\mathcal{A}_{\mathsf{real}}$;
  - Generate $s \in_R R$, send it to $\mathcal{A}_{\mathsf{real}}$ as from $M_c$.
  - $D_{\mathsf{sim}}[{+}{+}\ell_{D_{\mathsf{sim}}}] := s$.
- Receiving retrieve$(v)$ from $\mathcal{I}$:
  - Forward it, don't do anything else.
- Receiving $h_1 + h_2$ from $\mathcal{I}$:
  - Forward $h_1 + h_2$ to $\mathcal{A}_{\mathsf{real}}$.
  - $D_{\mathsf{sim}}[{+}{+}\ell_{D_{\mathsf{sim}}}] := D_{\mathsf{sim}}[h_1] + D_{\mathsf{sim}}[h_2]$.
  - (Send $D_{\mathsf{sim}}[\ell_{D_{\mathsf{sim}}}]$ to $\mathcal{A}_{\mathsf{real}}$ as from $M_c$.)

# Du-Atallah multiplication

- Let Alice have $a \in R$, Bob have $b \in R$.
- Alice, Bob and Charlie want to obtain $s_A, s_B, s_C \in R$, such that $s_A + s_B + s_C = a \cdot b$.
  - Party $X$ only learns $s_X$ and nothing else.
- Alice generates $\alpha_1 \in_R R$. Sends $\alpha_1$ to Charlie and $a + \alpha_1$ to Bob.
- Bob generates $\alpha_2 \in_R R$. Sends $\alpha_2$ to Charlie and $b + \alpha_2$ to Alice.
- The shares are defined as

$$
\begin{aligned}
s_A &= -\alpha_1(b + \alpha_2) \\
s_B &= b(a + \alpha_1) \\
s_C &= \alpha_1\alpha_2 \quad .
\end{aligned}
$$

  (**Exercise.** Verify that their sum is $a \cdot b$)
- Security: each of the parties only sends out uniformly randomly distributed values.

# Sharemind multiplication

- Let $v = s_0 + s_1 + s_2$ and $v' = s'_0 + s'_1 + s'_2$.

  $vv' = s_0s'_0 + s_0s'_1 + s_0s'_2 + s_1s'_0 + s_1s'_1 + s_1s'_2 + s_2s'_0 + s_2s'_1 + s_2s'_2$
- $M_i$ can compute $s_is'_i$ itself.
- To compute $s_is'_j$ we use Du-Atallah multiplication with $M_i$ as Alice, $M_j$ as Bob and $M_{3-i-j}$ as Charlie.
- Each party $M_i$ obtains six new shares from six instances of the Du-Atallah protocol.
- These six shares, as well as $s_is'_i$ are added together. The result is party $M_i$'s share of $vv'$.
- Finally, do resharing.
- Simulation:
  - Send a bunch of random values to the adversary.
  - Pick $D_{\mathsf{sim}}[{+}{+}\ell_{D_{\mathsf{sim}}}] \in_R R$.

# Share conversion

- Let $u \in \mathbb{Z}_2$ be shared as $u = u_0 \oplus u_1 \oplus u_2$.
- We want to get shares $s_0, s_1, s_2$, such that $u = s_0 + s_1 + s_2$ in $R$.
- Note that $u = u_0 + u_1 + u_2 - 2u_0u_1 - 2u_0u_2 - 2u_1u_2 + 4u_0u_1u_2$ in $R$.
- Compute this expression in distributed fashion:
  - $u_i$ will contribute to the share $s_i$ of $M_i$;
  - use Du-Atallah multiplication to get shares of $2u_iu_j$;
  - find shares of $4u_0u_1u_2$ :
    - let $M_2$ share $2u_2$ with the resharing protocol;
    - multiply $2u_0u_1$ and $2u_2$ with the multiplication protocol
  - Add the shares from the computation of all monomials;
  - Reshare.

# Bit extraction

- We have shares of the 32-bit value $u$.
- Let $u(k)$ be the $k$-th least significant bit of $u$. $u = \sum_{i=0}^{31} u(k)2^k$.
- We want to have shares of $u(0), \ldots, u(31)$ over $\mathbb{Z}_{2^{32}}$.

# Bit extraction

- We have shares of the 32-bit value $u$.
- Let $u(k)$ be the $k$-th least significant bit of $u$. $u = \sum_{i=0}^{31} u(k)2^k$.
- We want to have shares of $u(0), \ldots, u(31)$ over $\mathbb{Z}_{2^{32}}$.
- Let $M_i$ generate 32 random bits $r_i^0, \ldots, r_i^{31}$.
- We thus have shared 32 random bits $r^0, \ldots, r^{31}$ over $\mathbb{Z}_2$.
- Convert shares of $r^j$ to shares of $r(j) = r^j$ over $\mathbb{Z}_{2^{32}}$.
- Linearly combine shares of $r(0), \ldots, r(31)$ to get shares of $r$.
- Compute $a = u - r$ (linear combination). Publish $a$.
  - $a$ is distributed uniformly randomly; independently of $u$.
- Share the bits of $a$:
  - $a(j)_0 = a(j)$;
  - $a(j)_1 = a(j)_2 = 0$.
- We have shares of bits of $a$ and $r$, want to get shares of bits of $a + r$.

# Shares of bits of $u = a + r$

- Define $d(0) = a(0) + r(0)$, $d(i) = 2^i a(i) + 2^i r(i) + c(i)$ if $i > 0$.
  - $c(i)$ is the carry bit (see blackboard).
- $c(i) = 2^i \sum_{j=0}^{i-1} 2^j \cdot (a(j) + r(j) - u(j))$.
- $u(i)$ depends on $d(i)$ as follows:
  - We have $d(i) \in \{0, 2^i, 2^{i+1}, 2^{i+1} + 2^i\}$.
  - $u(i) = (d(i) \bmod 2^{i+1})/2^i$.
- Let $p(0), \ldots, p(31)$ be shared random bits.
- Let $f(i) = (d(i) + 2^i p(i)) \bmod 2^{i+1}$.
  - *modulo* is computed by each party.
  - $f(i) \in \{0, 2^i, 2^{i+1}, 2^{i+1} + 2^i, 2^{i+2}\}$
- Publish $f(i)$. If $f(i) \bmod 2^{i+1} = 2^i$ then $u(i) = 1 - p(i)$ else $u(i) = p(i)$.

# greater than

- Consider two values $v, v'$.
- We want to compute whether $v < v'$. Want to get the result as a shared bit.
- If $v, v' \in \mathbb{Z}_{2^{31}}$ then we can compute $v - v'$ and then check the sign bit.
    - sign bit $\equiv$ most significant bit
- Sign bit is given by bit extraction.