Preprocessing Based Verification of Multiparty Protocols with an Honest Majority

20.07.17

Peeter Laud  Alisa Pankova  Roman Jagomägis
Secure Multiparty Computation

\[(y_a, y_b, y_c) = f(x_a, x_b, x_c)\]
Secure Multiparty Computation

\( (y_a, y_b, y_c) = f(x_a, x_b, x_c) \)

- ALICE: \( x_a \) and \( y_a = 'yes' \)
- BOB: \( x_b \) and \( y_b = 'yes' \)
- CHRIS: \( x_c \) and \( y_c = 'yes' \)
Secure Multiparty Computation

\[(y_a, y_b, y_c) = f(x_a, x_b, x_c)\]

- Passive adversary: all parties follow the protocol.
- Active adversary: corrupted parties may cheat.
- Covert adversary: will not cheat if it will be caught.

ALICE

\[x_a = \text{[files]} \quad y_a = 'y'\]

BOB

\[x_b = \text{[files]} \quad y_b = 'e'\]

CHRIS

\[x_c = \text{[files]} \quad y_c = 's'\]
Secure Multiparty Computation

\[(y_a, y_b, y_c) = f(x_a, x_b, x_c)\]
Secure Multiparty Computation

- Passive adversary: all parties follow the protocol.
Secure Multiparty Computation

- Passive adversary: all parties follow the protocol.
- Active adversary: corrupted parties may cheat.
Secure Multiparty Computation

- Passive adversary: all parties follow the protocol.
- Active adversary: corrupted parties may cheat.
Secure Multiparty Computation

- Passive adversary: all parties follow the protocol.
- Active adversary: corrupted parties may cheat.
Secure Multiparty Computation

- **Passive adversary**: all parties follow the protocol.
- **Active adversary**: corrupted parties may cheat.
Secure Multiparty Computation

- **Passive adversary:** all parties follow the protocol.
- **Active adversary:** corrupted parties may cheat.
- **Covert adversary:** will not cheat if it will be caught.
Verifiable MPC with Honest Majority

- **Execution**: run the passively secure protocol.
Verifiable MPC with Honest Majority

- **Execution**: run the passively secure protocol.
- **Verification**: each party proves that it followed the protocol.

B = "Alice has cheated, Bob and Chris are honest"
Verifiable MPC with Honest Majority

- **Preprocessing**: generate correlated randomness.
- **Execution**: run the passively secure protocol.
- **Verification**: each party proves that it followed the protocol.

\[ B = "Alice \text{ has cheated, Bob and Chris are honest}" \]
Verifiable MPC with Honest Majority

- **Preprocessing:** generate correlated randomness.
- **Execution:** run the passively secure protocol.
- **Verification:** each party proves that it followed the protocol.

B = "Alice has cheated, Bob and Chris are honest"
Execution Phase

- Run the initial passively secure protocol.
- Each message $m$ is provided with a sender’s signature $\sigma_m$. 
Execution Phase

- Run the initial passively secure protocol.
- Each message $m$ is provided with a sender’s signature $\sigma_m$.

![Diagram showing Alice sending a message $m, \sigma_m$ to Bob, with Bob saying, “I have not received $m, \sigma_m$.”]
Execution Phase

- Run the initial passively secure protocol.
- Each message $m$ is provided with a sender’s signature $\sigma_m$.

I do have sent $m, \sigma_m$

ALICE

$m, \sigma_m$

BOB

I have not received $m, \sigma_m$
Execution Phase

- Run the initial passively secure protocol.
- Each message $m$ is provided with a sender’s signature $\sigma_m$.

- If Alice refuses to send $(m, \sigma_m)$ Bob asks Chris to deliver it.
- If Alice or Bob is corrupt, $(m, \sigma_m)$ is already known to the attacker anyway.
Verification phase

Each party (the prover \( P \)) proves its honesty to the other parties (the verifiers \( V_1 \) and \( V_2 \)).

All relevant values of \( P \) are shared among \( V_1 \) and \( V_2 \):

- **Message** \( m \): \( m + 0 \) or \( 0 + m \)
- **Input** \( x \): \( x_1 + x_2 \)
- **Correlated randomness** \( r \): \( r_1 + r_2 \)

known by \( P \), shared in the preprocessing phase.

All shares are signed by the prover.
Verification phase (reproducing computation of $P$)

$P$ takes precomputed correlated randomness (e.g. Beaver triples $\langle a, b, c \rangle$ s.t. $c = a \cdot b$).

$P$ sends hints to $V_1$ and $V_2$.

$V_1$ and $V_2$ use the hints to reproduce computation of $P$.

$V_1$ and $V_2$ verify the hints.

$V_1$ and $V_2$ check if they get committed messages of $P$. 

$(x, y, z), \quad z = x \cdot y$

$(x_1, y_1, z_1)$

$(x_2, y_2, z_2)$
Verification phase (reproducing computation of $P$)

- $P$ takes precomputed correlated randomness (e.g. Beaver triples $(a, b, c)$ s.t. $c = a \cdot b$).

$P$ sends hints to $V_1$ and $V_2$.

$V_1$ and $V_2$ use the hints to reproduce computation of $P$.

$V_1$ and $V_2$ verify the hints.

$V_1$ and $V_2$ check if they get committed messages of $P$. 
Verification phase (reproducing computation of $P$)

- $P$ takes precomputed correlated randomness (e.g. Beaver triples $(a, b, c)$ s.t. $c = a \cdot b$).
- $P$ sends hints to $V_1$ and $V_2$.

\[
\begin{align*}
    \mathbf{x}' &= (x - a) \\
    \mathbf{y}' &= (y - b)
\end{align*}
\]

\[
\begin{align*}
    (x, y, z), & \quad z = x \cdot y \\
    (a, b, c), & \quad c = a \cdot b
\end{align*}
\]
Verification phase (reproducing computation of $P$)

- $P$ takes precomputed correlated randomness (e.g. Beaver triples $(a, b, c)$ s.t. $c = a \cdot b$).
- $P$ sends hints to $V_1$ and $V_2$.
- $V_1$ and $V_2$ use the hints to reproduce computation of $P$.

$\begin{align*}
  x' &= (x - a) \\
  y' &= (y - b)
\end{align*}$

$\begin{align*}
  (x_1, y_1, z_1) \\
  (a_1, b_1, c_1)
\end{align*}$

$\begin{align*}
  v_1 &= x'a_1 + y'b_1 + x'y' + c_1
\end{align*}$

$\begin{align*}
  (x_2, y_2, z_2) \\
  (a_2, b_2, c_2)
\end{align*}$

$\begin{align*}
  v_2 &= x'a_2 + y'b_2 + x'y' + c_2
\end{align*}$
Verification phase (reproducing computation of $P$)

- $P$ takes precomputed correlated randomness (e.g. Beaver triples $(a, b, c)$ s.t. $c = a \cdot b$).
- $P$ sends hints to $V_1$ and $V_2$.
- $V_1$ and $V_2$ use the hints to reproduce computation of $P$.
- $V_1$ and $V_2$ verify the hints.

\[
\begin{align*}
(x, y, z), & \quad z = x \cdot y \\
(a, b, c), & \quad c = a \cdot b
\end{align*}
\]
Verification phase (reproducing computation of $P$)

- $P$ takes precomputed correlated randomness (e.g. Beaver triples $(a, b, c)$ s.t. $c = a \cdot b$).
- $P$ sends hints to $V_1$ and $V_2$.
- $V_1$ and $V_2$ use the hints to reproduce computation of $P$.
- $V_1$ and $V_2$ verify the hints.
- $V_1$ and $V_2$ check if they get committed messages of $P$.

\[
\begin{align*}
(x, y, z), & \quad z = x \cdot y \\
(a, b, c), & \quad c = a \cdot b
\end{align*}
\]

\[
\begin{align*}
x' &= (x - a) \\
y' &= (y - b)
\end{align*}
\]

\[
\begin{align*}
(x_1, y_1, z_1) & \quad (x_2, y_2, z_2) \\
(a_1, b_1, c_1) & \quad (a_2, b_2, c_2)
\end{align*}
\]

\[
\begin{align*}
v_1 = x' a_1 + y' b_1 + x'y' + c_1 & \quad v_2 = x' a_2 + y' b_2 + x'y' + c_2 \\
& \quad v - z = 0
\end{align*}
\]
Verification phase (reproducing computation of $P$)

- $P$ takes precomputed correlated randomness (e.g. Beaver triples $(a, b, c)$ s.t. $c = a \cdot b$).
- $P$ sends hints to $V_1$ and $V_2$.
- $V_1$ and $V_2$ use the hints to reproduce computation of $P$.
- $V_1$ and $V_2$ verify the hints.
- $V_1$ and $V_2$ check if they get committed messages of $P$.

$(x, y, z), \quad z = x \cdot y$

$(a, b, c), \quad c = a \cdot b$

$$
\begin{align*}
  x' &= (x - a) \\
  y' &= (y - b)
\end{align*}
$$

$(x_1, y_1, z_1)$

$(a_1, b_1, c_1)$

$v_1 = x'a_1 + y'b_1 + x'y' + c_1$

$x' - x' = 0$

$y' - y' = 0$

$y - b - y' = 0$

$x - a - x' = 0$

$(x_2, y_2, z_2)$

$(a_2, b_2, c_2)$

$v_2 = x'a_2 + y'b_2 + x'y' + c_2$

$v - z = 0$
Verification phase (checking if $z = 0$)

- $V_1$ and $V_2$ exchange $h_1 = H(z_1)$ and $h_2 = H(-z_2)$, and check $h_1 = h_2$.
Verification phase (checking if $z = 0$)

- $V_1$ and $V_2$ exchange $h_1 = H(z_1)$ and $h_2 = H(-z_2)$, and check $h_1 = h_2$.
- If $h_1 \neq h_2$, they send $h_1$ and $h_2$ to $P$. 
Verification phase (checking if $z = 0$)

- $V_1$ and $V_2$ exchange $h_1 = H(z_1)$ and $h_2 = H(-z_2)$, and check $h_1 = h_2$.
- If $h_1 \neq h_2$, they send $h_1$ and $h_2$ to $P$.
- $P$ has right to complain against one verifier (e.g. $V_1$).
Verification phase (checking if $z = 0$)

- $V_1$ and $V_2$ exchange $h_1 = H(z_1)$ and $h_2 = H(-z_2)$, and check $h_1 = h_2$.
- If $h_1 \neq h_2$, they send $h_1$ and $h_2$ to $P$.
- $P$ has right to complain against one verifier (e.g. $V_1$).
- $V_1$ opens its shares of $P$ commitments with all signatures.
Verification phase (checking if $z = 0$)

- $V_1$ and $V_2$ exchange $h_1 = H(z_1)$ and $h_2 = H(-z_2)$, and check $h_1 = h_2$.
- If $h_1 \neq h_2$, they send $h_1$ and $h_2$ to $P$.
- $P$ has right to complain against one verifier (e.g. $V_1$).
- $V_1$ opens its shares of $P$ commitments with all signatures.
- $V_2$ repeats the computation of $V_1$, getting $h_1$. 

![Diagram showing the process of verification phase]
Preprocessing Phase

- The prover $P$ generates correlated randomness (e.g., Beaver triples in a certain ring $\mathbb{Z}_m$).
Preprocessing Phase

- The prover $P$ generates correlated randomness (e.g. Beaver triples in a certain ring $\mathbb{Z}_m$).
- It additively shares the randomness among $V_1$ and $V_2$.
Preprocessing Phase

- The prover $P$ generates correlated randomness (e.g., Beaver triples in a certain ring $\mathbb{Z}_m$).
- It additively shares the randomness among $V_1$ and $V_2$.
- $V_1$ and $V_2$ run cut-and-choose and pairwise checks to verify that correlation holds (e.g., that $a \cdot b = c$).
Preprocessing Phase

- The prover $P$ generates correlated randomness (e.g. Beaver triples in a certain ring $\mathbb{Z}_m$).
- It additively shares the randomness among $V_1$ and $V_2$.
- $V_1$ and $V_2$ run cut-and-choose and pairwise checks to verify that correlation holds (e.g. that $a \cdot b = c$).

\[ c = a \cdot b \]
\[ c' = a' \cdot b' \]

\[ (a_1, b_1, c_1) \]
\[ (a_1', b_1', c_1') \]
\[ \ldots \]

\[ (a_2, b_2, c_2) \]
\[ (a_2', b_2', c_2') \]
\[ \ldots \]

cut-and-choose: open and verify some triples

pairwise check
\[ (a - a')b + (b - b')a' - c + c' = 0 \]
Preprocessing Phase (other preprocessed tuples)

- We also have other types of preprocessed tuples:
  - Trusted bits $b \in \{0, 1\}$ shared over $\mathbb{Z}_{2m}$.
  - Characteristic vector tuple $(r, \vec{b})$ (i.e. $b_r = 0$ iff $i \neq r$).
  - Rotation tuple $(r, \vec{a}, \vec{b})$ s.t the vector $\vec{b}$ is $\vec{a}$ rotated by $r$.
  - Permutation tuple $(\pi, \vec{a}, \vec{b})$ s.t $\vec{b} = \pi(\vec{a})$.

- Their generation and verification is analogous.
Summary

- We proposed a generic method for achieving covert security under honest majority assumption.
- Applying it to Sharemind SMC platform, we get efficient actively secure protocols with identifiable abort.
- The overhead of the execution phase is insignificant.
- In practice, the bottleneck of active security is generation of preprocessed tuples.