Precomputed Verification of Multiparty Protocols with Honest Majority

07.02.15

Peeter Laud    Alisa Pankova
Three good friends worked hard to collect some wealth.

The wealth was stored in a stash locked with three keys.

No one could open it unless having all the three keys.

Once upon a time, they decided to check if they can already buy a house.
With years, the friends did not trust each other anymore.
They solved the problem using SMC.

The computation was represented by arithmetic circuits.
Only the output was published.
The friends executed a passively secure protocol.
But malicious Alice deviated from the rules.
The friends executed a passively secure protocol.
But malicious Alice deviated from the rules.

By the end of the computation, Alice has got all the keys!
Alice has stolen the entire treasure.
Alice has stolen the entire treasure.

The friends were advised to use ZK proofs next time.
Time Problems

- Now the three friends work together again.
- Soon, they will count their wealth.
- Bob and Chris propose to use ZK proofs in the end.
- But Alice says that she will have no time for that.

However, there is still time until the counting.
Formalizing the Computation

- Each party’s input is a vector over a ring $R$.
- Each communication is a vector over $R$.

If the input and the communication was committed, everyone could re-compute the circuits.

But these values are not public!
Commitments

- $a, ab, ba, ac, ca$ should be committed.
- Let:

\[
\begin{align*}
a &= a_1 + a_2, \\
ab &= ab_1 + ab_2, \\
ba &= ba_1 + ba_2, \\
ac &= ac_1 + ac_2, \\
ca &= ca_1 + ca_2,
\end{align*}
\]
Consistency

- Do the committed $ab_i, ba_i (ac_i, ca_i)$ indeed correspond to $ab, ba (ac, ca)$ that Bob (Chris) has actually seen?
Consistency

- Do the committed $ab_i, ba_i (ac_i, ca_i)$ indeed correspond to $ab, ba (ac, ca)$ that Bob (Chris) has actually seen?
- In the initial protocol, each message is signed.
Consistency

- Do the committed $ab_i$, $ba_i$ ($ac_i$, $ca_i$) indeed correspond to $ab$, $ba$ ($ac$, $ca$) that Bob (Chris) has actually seen?
- In the initial protocol, each message is signed.
- If a malicious sender refuses to sign the message, the receiver asks the third party to deliver it.
Receiver’s approval: malicious sender

- The sharing is done in the end of the protocol.
- Each message is shared by the sender and approved by the receiver.

\[
Sign_A(ab) \\
ab \overset{?}{=} ab_1 + ab_2
\]

If Bob sees that something is wrong, or Alice is silent, it publishes \(Sign_A(ab)\).
Alice cannot disprove Bob’s shares since it needs $\text{Sign}_B(ba^*)$.

If Alice does not respond, Chris may ask for $ba^*_2$ directly from Bob.
What we have:

- The input shares $a_i$ can be committed before the protocol starts.
- The communication shares $ab_i$, $ba_i$, $ac_i$, $ca_i$ are committed after the execution.
- The verifiers want to simulate protocol execution as locally as possible.
- Only addition can be done locally, but not multiplication.
Preprocessing

- The parties still have enough time before the protocol run starts.
- A Beaver triple is $(r_x, r_y, r_{xy})$ s.t $r_{xy} = r_x \cdot r_y$.
- $r_x, r_y \$ R$, compute $r_{xy} := r_x \cdot r_y$.
- $r_x, r_y, r_{xy}$ are shared amongst Bob and Chris.
- $r_x, r_y, r_{xy}$ are all sent to Alice.

- Triple generation can be secure against active adversary.
Verification

- Each addition can be done locally.
- If it was $x = (x' + r_x)$ and $y = (y' + r_y)$, 
  
  $x \ast y = x'y' + x'r_y + y'r_x + r_{xy}$.

- Alice knows $(r_x, r_y, r_{xy})$.

- She is asked to publish $x'$ and $y'$ such that 
  
  $x = (x' + r_x)$ and $y = (y' + r_y)$.
The parties compute the shares
\[ \tilde{z}^i = x'y' + x'r_y^i + y'r_x^i + r_{xy}^i \]
locally.

But \( x, y, z \) may have already been committed.

Is it true that \( x = (x' + r_x), y = (y' + r_y), z = \tilde{z} \)?

Since the sharing can be different, it can be verified only online.

This check is delayed.
In the end

- It should be checked online if the locally computed result is indeed the committed \( ab \) and \( ac \).
- Each online check is of the form \( x - y = 0 \).
- Let \( d = d^1 + d^2 \) be the vector of all such checks.
- We need to check \( d^1 + d^2 = 0 \), or \( d^1 = -d^2 \).
- Compute and publish \( h^1 = h(d^1) \) and \( h^2 = h(-d^2) \).
- Check \( h^1 = h^2 \).
Protecting Honest Provers

- A proof of honesty is required from each party.

$$\pi_1 = \begin{align*} b_1 \\ ab_1 \\ ba_1 \\ bc_1 \\ cb_1 \end{align*}$$

$$\pi_2 = \begin{align*} b_2 \\ ab_2 \\ ba_2 \\ bc_2 \\ cb_2 \end{align*}$$

Bob can compute $h_1$ and $h_2$ himself. It may publish $h_2$ if it is wrong. If Alice cannot disprove it, Chris accepts Bob's proof.
A proof of honesty is required from each party.

Bob can compute $h^1$ and $h^2$ himself.

It may publish $h^2$ himself if it is wrong.

If Alice cannot disprove it, Chris accepts Bob’s proof.
Computation over Multiple Rings

Transitions between rings:
- From larger to smaller: $z = trunc(x)$.
- From smaller to larger: $z = zext(x)$.

Let $x \in \mathbb{Z}_{2^k}$, $z \in \mathbb{Z}_{2^\ell}$, $k < \ell$.

The shares of $z = trunc(x)$ can be computed locally.

Two additional online checks are needed for each $z = zext(x)$:
- $x = trunc(z)$;
- $z = z_0 + z_1 \cdot 2 + \ldots + z_{k-1} \cdot 2^{k-1}$.

This works only if $z_i$ are bits.
Random Bit Shares

- Generate a random bit share $r_b$ for a bit variable $b$.
- After the execution, publish $b'$ s.t. $b = (b' + r_b) \mod 2$.
- However, the shares may be not in $\mathbb{Z}_2$.
  - If $b' = 0$, take the share $b^i = r^i_b$.
  - If $b' = 1$, take the share $b^i = 1 - r^i_b$.

Bit generation can be secure against active adversary.
Conclusion

- The verification time can be decreased by a longer preprocessing phase.
- It is easy to generalize verification to several rings.
- The method can be extended to $n$ parties.
And we still have *not protected* the secrets *from honest users!* Some other security model would be preferable.