Secure Computation with Low Communication from Cross-checking

Dov Gordon (George Mason U.)
Samuel Ranellucci (Unbound Tech)
Xiao Wang (MIT & Boston U.)

Secure Computation

- 4 parties each hold private data.
- They wish to compute $C(x_1, x_2, x_3, x_4)$
- Nobody should learn anything more than the output.
- We assume honest majority: at most 1 malicious actor.
  - The adversary can behave arbitrarily.

Why 4PC?

- Existing protocols support n parties, with n-1 maliciously colluding.
- Weaker assumptions lead to better performance!
- As MPC has become more practical, a common use-case that appears is one of out-sourced computation:
  - many parties secret share their data among a few computing servers.
  - This has most often been done with 3 servers, because the honest majority assumption leads to more efficient protocols.

Results

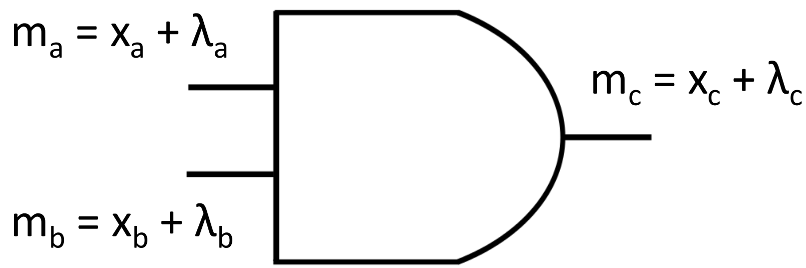
- We provide a 4-party protocol requiring just $6|C|\log|F| + O(\kappa)$ total communication.
- We can compute over arbitrary fields, including GF_2 (Boolean circuits).
- We can even compute over arbitrary rings, such as $GF_{2^{32}}$.
- We demonstrate a robust variant of our protocol, guaranteeing output.

Related Work

- Best 2 party protocols require about 2300 bits of communication per gate [1,2].
- Furukawa et al. demonstrate a protocol in the 3-party setting that requires 21 bits of communication per gate [3].

[1] Wang et al. Authenticated garbling and efficient maliciously secure 2-party computation, 2017
[2] Nielsen et al. A new approach to practical active-secure two-party computation, 2012.
[3] Furukawa et al. High-throughput secure three-party computation for malicious adversaries and an honest majority, 2017.

Masked Evaluations

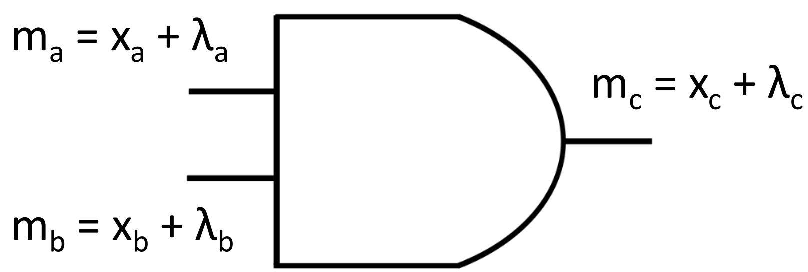


2 parties hold:
masked input wire values, m_a and m_b, and
secret shares of $\lambda_a, \lambda_b, \lambda_c$ and $\lambda_a\lambda_b$.
They compute masked output m_c.

$$m_a \cdot m_b - m_a\cdot\langle\lambda_b\rangle - m_b\cdot\langle\lambda_a\rangle + \langle\lambda_a\lambda_b\rangle + \langle\lambda_c\rangle =$$
$$[(x_a+\lambda_a)(x_b+\lambda_b) - m_a\cdot\langle\lambda_b\rangle - m_b\cdot\langle\lambda_a\rangle] + \langle\lambda_a\lambda_b\rangle + \langle\lambda_c\rangle =$$
$$[\langle x_a x_b - \lambda_a\lambda_b\rangle] + \langle\lambda_c\rangle + \langle\lambda_a\lambda_b\rangle = \langle x_a x_b + \lambda_c\rangle$$

The parties open their shares to obtain m_c.
Communication cost: 4|C|

Masked Evaluations



2 parties hold:
masked input wire values, m_a and m_b, and
secret shares of λ_a, λ_b, λ_c and $\lambda_a\lambda_b$.
They compute masked output m_c.

Beaver triples, but we open shares of a blinded product.

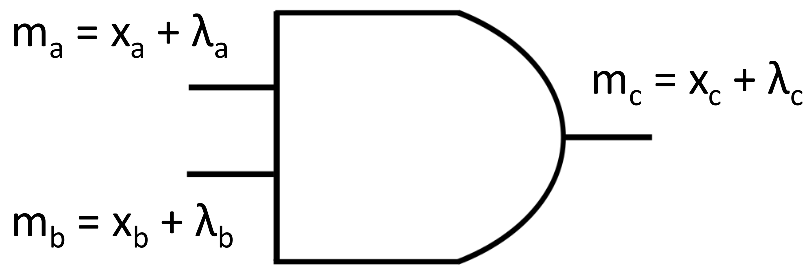
$$m_a \cdot m_b - m_a\cdot\langle\lambda_b\rangle - m_b\cdot\langle\lambda_a\rangle + \langle\lambda_a\lambda_b\rangle + \langle\lambda_c\rangle =$$

$$[(x_a+\lambda_a)(x_b+\lambda_b) - m_a\cdot\langle\lambda_b\rangle - m_b\cdot\langle\lambda_a\rangle] + \langle\lambda_a\lambda_b\rangle + \langle\lambda_c\rangle =$$

$$[\langle x_a x_b - \lambda_a\lambda_b\rangle] + \langle\lambda_c\rangle + \langle\lambda_a\lambda_b\rangle = \langle x_a x_b + \lambda_c\rangle$$

The parties open their shares to obtain m_c.

Masked Evaluations



2 parties hold:
masked input wire values, m_a and m_b, and
secret shares of λ_a, λ_b, λ_c and $\lambda_a\lambda_b$.
They compute masked output m_c.

$m_a \cdot m_b - m_a\cdot\langle\lambda_b\rangle - m_b\cdot\langle\lambda_a\rangle + \langle\lambda_a\lambda_b\rangle + \langle\lambda_c\rangle \quad =$

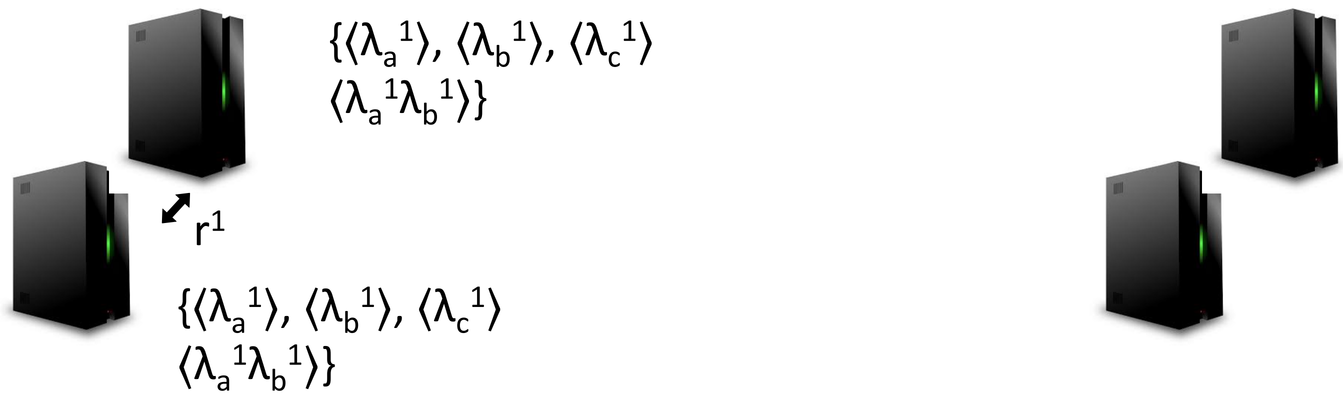
$[(x_a+\lambda_a)(x_b+\lambda_b) - m_a\cdot\langle\lambda_b\rangle - m_b\cdot\langle\lambda_a\rangle] + \langle\lambda_a\lambda_b\rangle + \langle\lambda_c\rangle \quad =$

$[\langle x_a x_b - \lambda_a\lambda_b\rangle] + \langle\lambda_c\rangle + \langle\lambda_a\lambda_b\rangle \quad = \langle x_a x_b + \lambda_c\rangle$

The parties open their shares to obtain m_c.

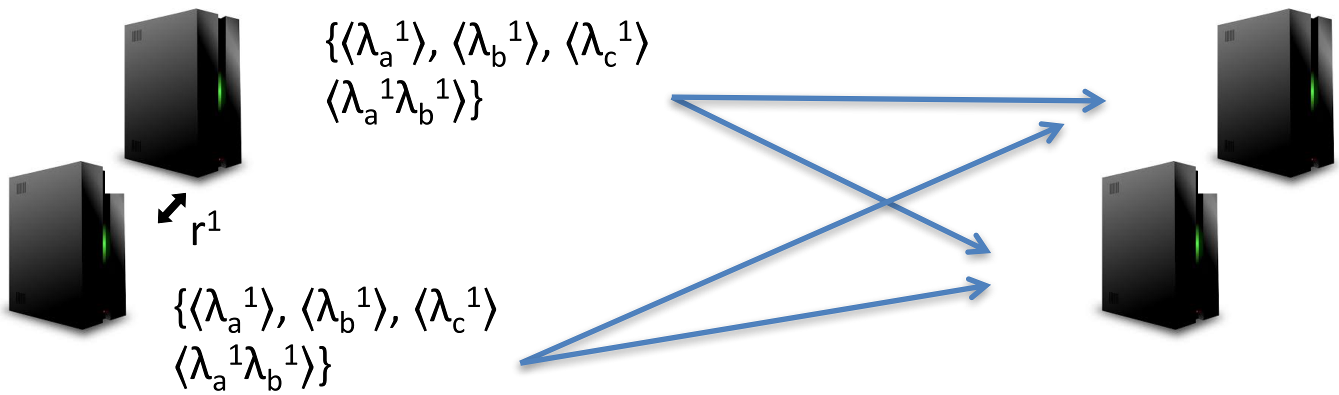
Adversary can add arbitrary value to m_c.

Preprocessing



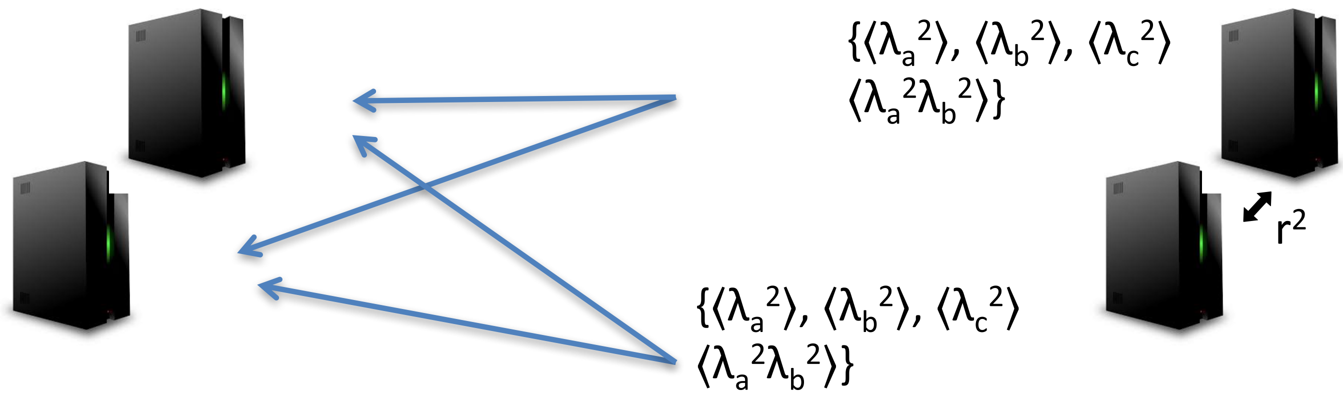
- One pair of parties creates 2 identical copies of the preprocessing for the other pair to use.

Preprocessing



- One pair of parties creates 2 identical copies of the preprocessing for the other pair to use.
- They both send the shares to the other pair, who abort if the copies aren't identical.

Preprocessing



- The 2nd pair does the same with their own shared randomness.
- Each pair will execute its own computation, using the preprocessing provided by the other pair.
- Communication: $2|C| + 6\kappa$.

Cross Checking

$$m_w^2 + \lambda_w^1 \stackrel{?}{=} m_w^1 + \lambda_w^2$$

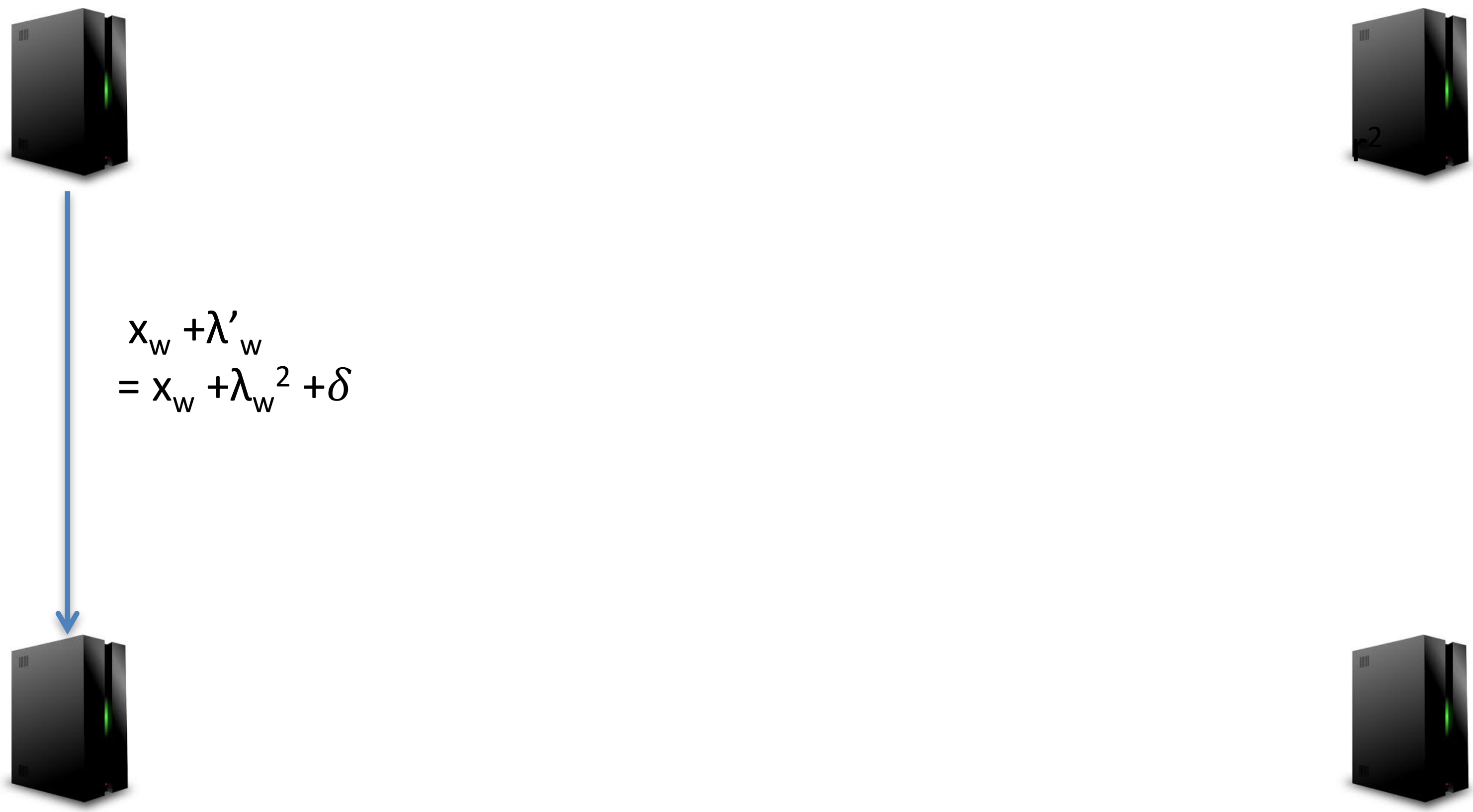
$$x_w + \lambda_w^2 + \lambda_w^1 \stackrel{?}{=} x_w + \lambda_w^1 + \lambda_w^2$$

However, the comparison requires care.

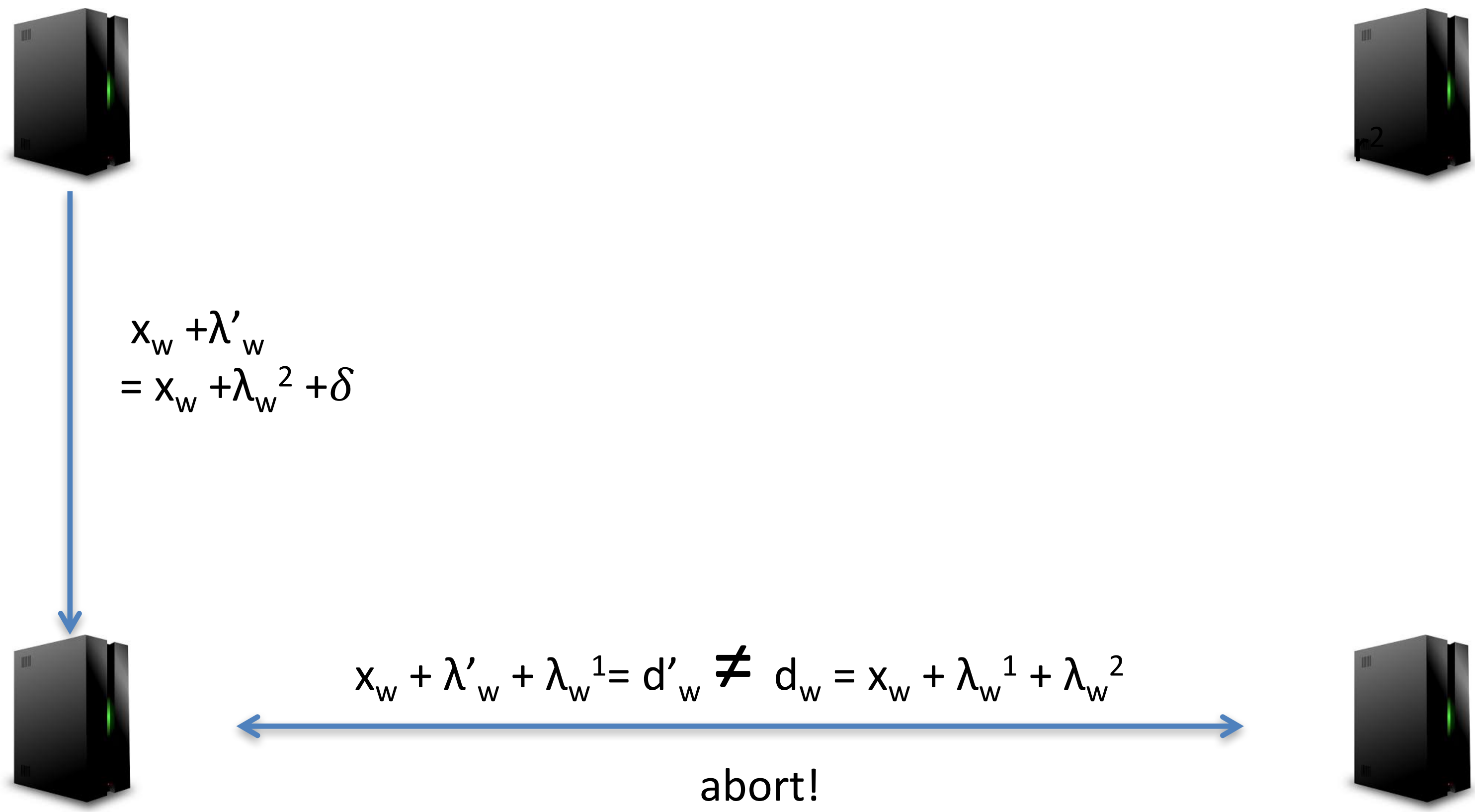
Consider this insecure protocol:

1. The pairs evaluate the full circuit, each pair recovering all doubly-masked values, $\{d_w\}$.
2. P1 and P3 compare their values, abort on an inconsistency.
3. P2 and P4 compare their values, abort on an inconsistency.

Cross Checking



Cross Checking



Cross Checking

$d'_w - \delta = d_w$

continue!

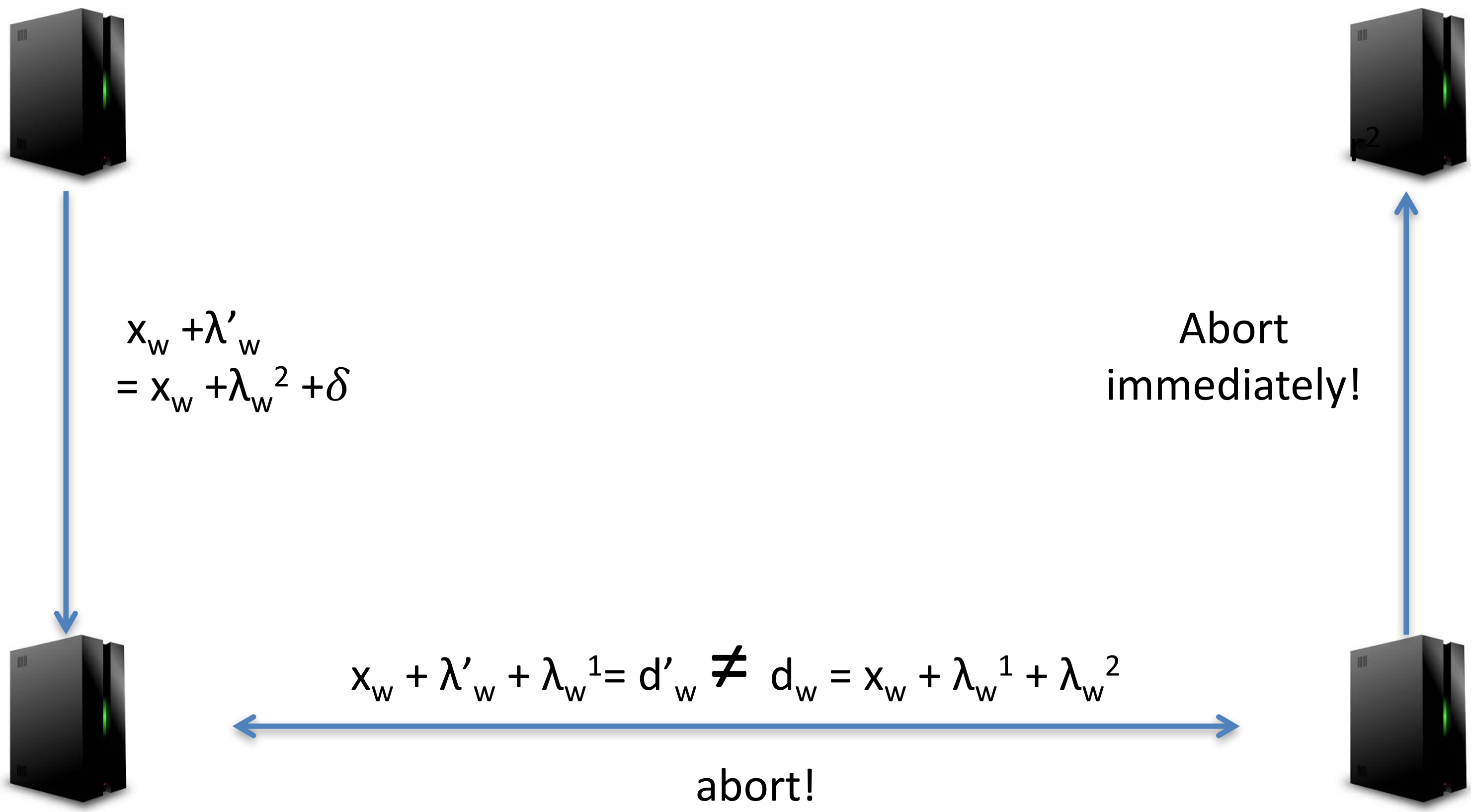
After adding δ on one wire, but correcting all $\{d_w\}$ values so that the cross check passes:

for any wire y dependent on w, the value $d'_y - d_y$ leaks information about the input.

$x_w + \lambda'_w + \lambda_w^2 = d'_w \neq d_w = x_w + \lambda_w^1 + \lambda_w^2$

abort!

Cross Checking



Cross Checking

(better communication)

Cross checking is secure if we go wire by wire.

We don’t want to send a field element for every wire during cross checking.

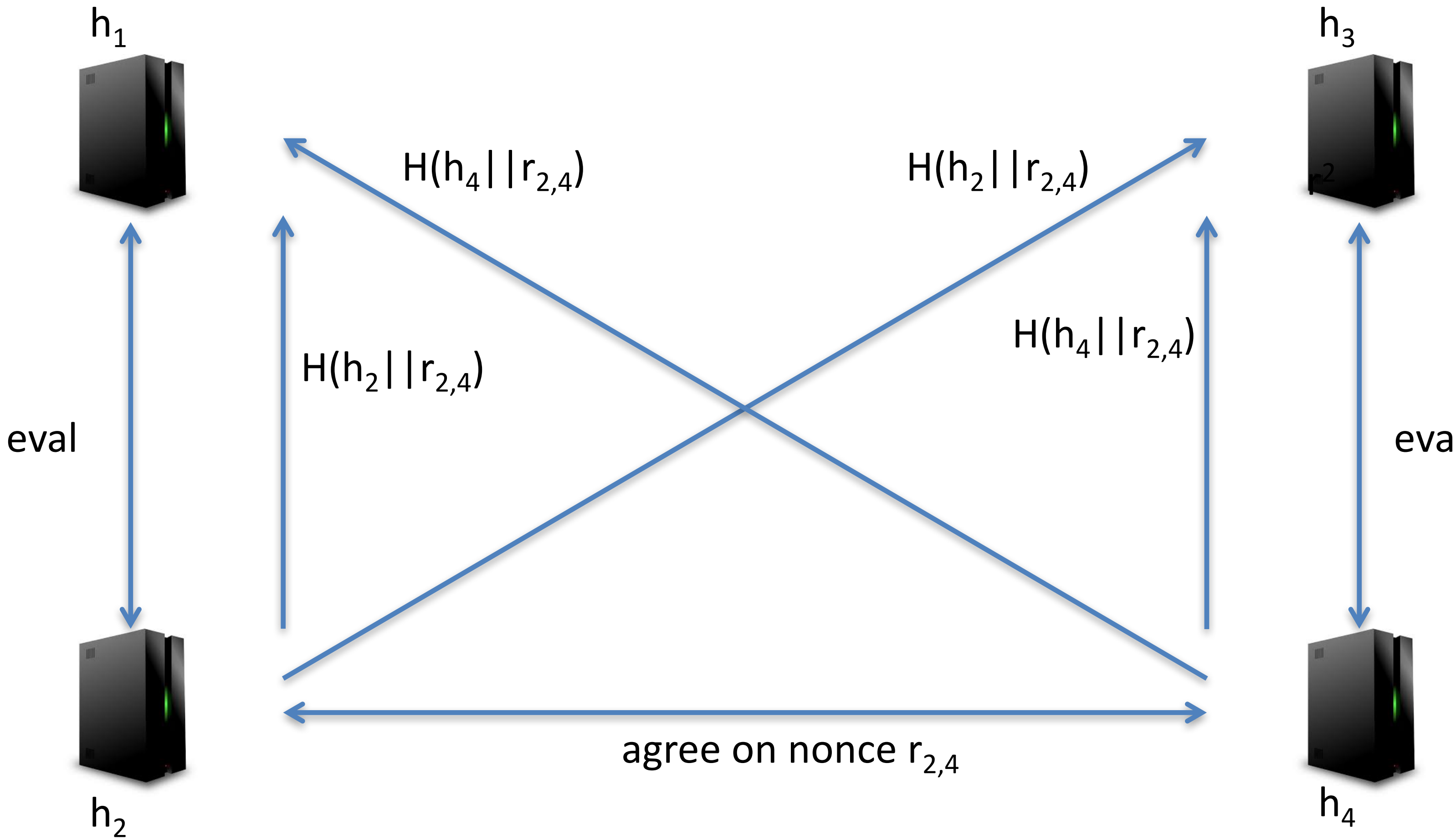
Instead:

1. Each pair computes all of their $\{d_w\}$ values.
2. Each computes $H(d_1, . . , d_C)$.
3. Evaluate a (generic) 4pc:
   $F(h_1, h_2, h_3, h_4) = 1 \leftrightarrow h_1 = h_3 \wedge h_2 = h_4$

Communication cost: poly(κ) (depends on 4pc protocol)

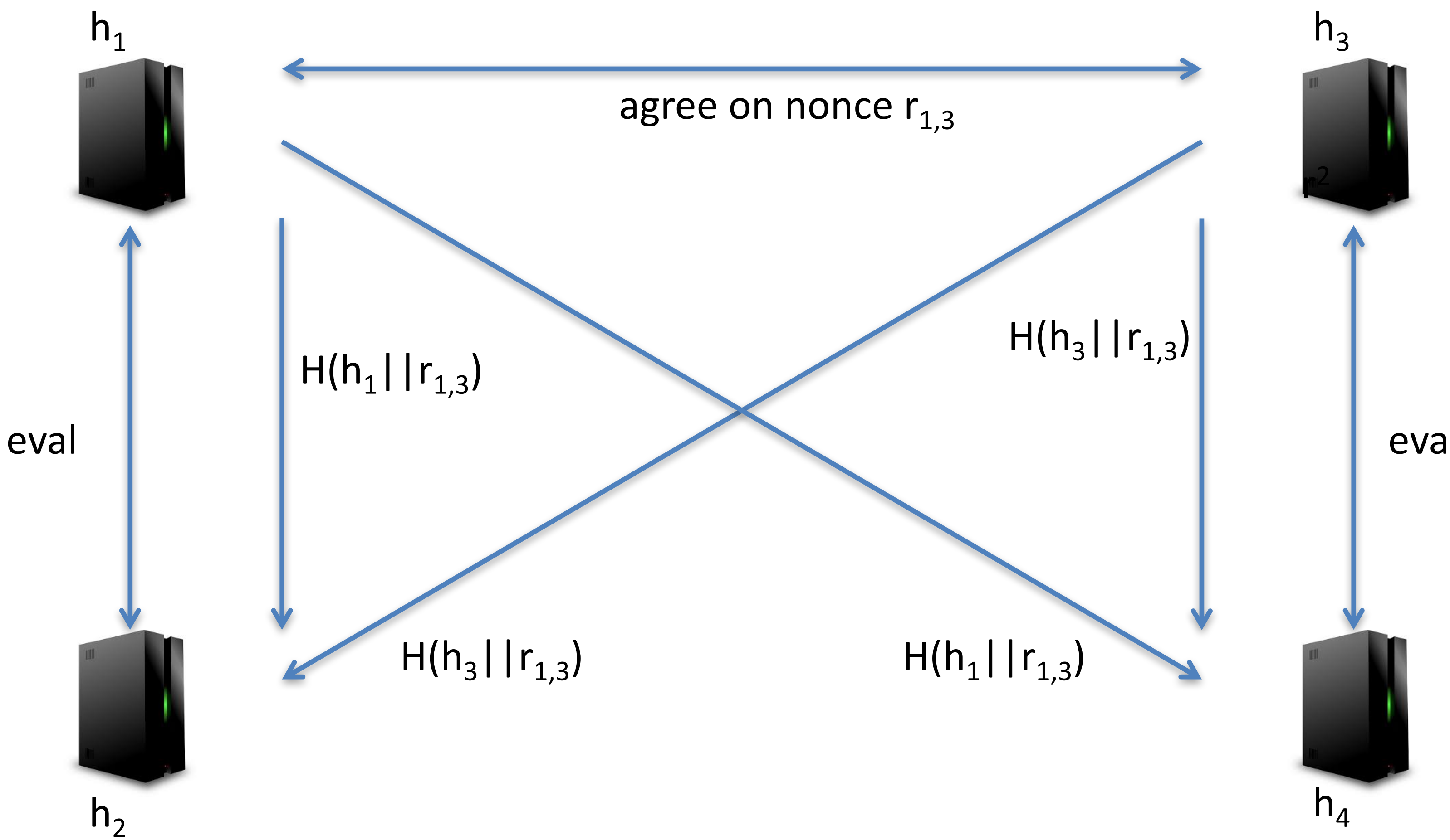
Cross Checking

(still better communication)



Cross Checking

(still better communication)



Cross Checking

(still better communication)

h_1

If $H(h_2||r_{2,4}) \neq H(h_4||r_{2,4})$
$veto_1 = 1$

h_3

If $H(h_2||r_{2,4}) \neq H(h_4||r_{2,4})$
$veto_3 = 1$

Securely compute 3 OR gates:
$veto_1 \vee veto_2 \vee veto_3 \vee veto_4$
Recall: gate by gate cross checking is secure!

If $H(h_1||r_{1,3}) \neq H(h_3||r_{1,3})$
$veto_2 = 1$

If $H(h_1||r_{1,3}) \neq H(h_3||r_{1,3})$
$veto_4 = 1$

h_4

h_2

Cross Checking
(still better communication)

h_1

If $H(h_2||r_{2,4}) \neq H(h_4||r_{2,4})$
$veto_1 = 1$

h_3

If $H(h_2||r_{2,4}) \neq H(h_4||r_{2,4})$
$veto_3 = 1$

Securely compute 3 OR gates:
$veto_1 \vee veto_2 \vee veto_3 \vee veto_4$
Recall: gate by gate cross checking is secure!

If $H(h_1||r_{1,3}) \neq H(h_3||r_{1,3})$
$veto_2 = 1$

If $H(h_1||r_{1,3}) \neq H(h_3||r_{1,3})$
$veto_4 = 1$

h_4

h_2

Communication cost: about 10κ

Robustness

- Robust Preprocessing
  - Using committing encryption, broadcast, and signatures, can agree on who was inconsistent.
  - One exception: say P1 sent nothing to P3.
    - P3 can't prove that P1 was malicious, rather than him.
    - However, he can ignore P1, and use the preprocessing of P2, knowing it is honestly generated.
- Robust input sharing
  - straightforward, using broadcast and signatures.

Robustness

- Robust cross checking
  - Go back to checking gate by gate.
  - Say P_3 reports an inconsistency. 3 possible reasons:
    - The masked eval. performed by P_1 and P_2 is invalid.
    - The masked eval. performed by P_3 and P_4 is invalid.
    - Both evaluations were executed correctly, but either P_1 modified his reported masked evaluation, or P_3 complained for no valid reason.

THANKS!