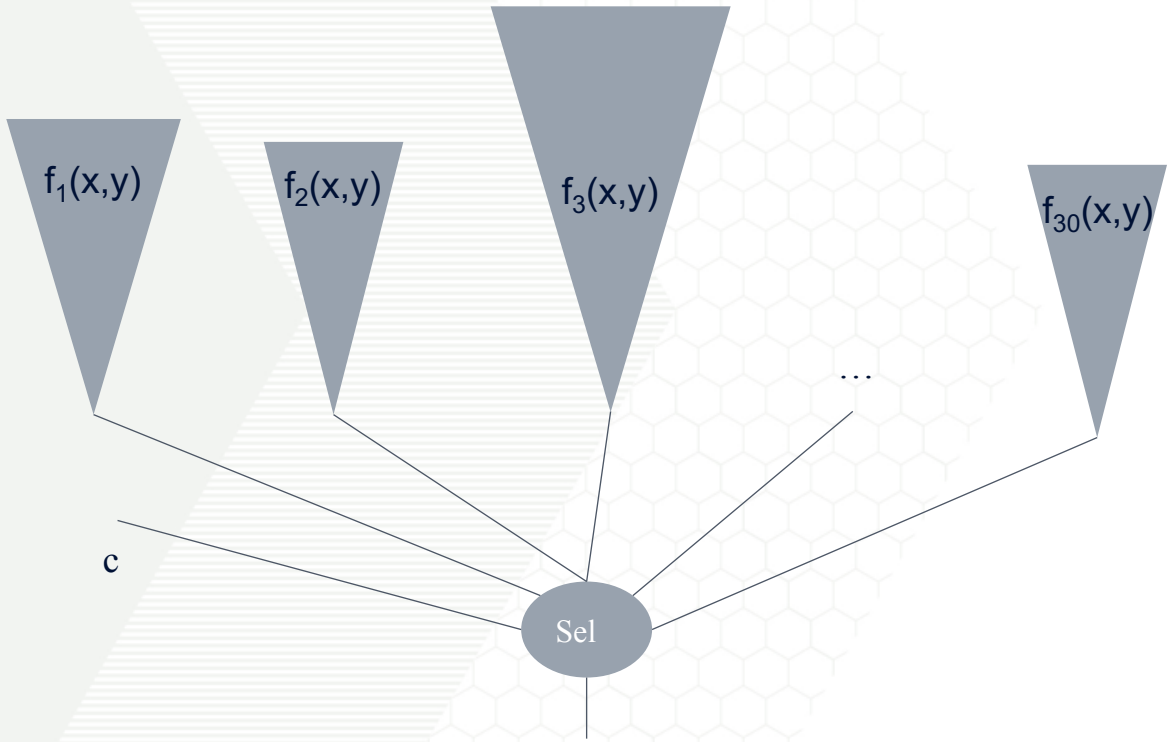
# FREE IF:
# HOW TO OMIT INACTIVE BRANCHES AND IMPLEMENT S-UNIVERSAL GARBLED CIRCUIT ALMOST FOR FREE

VLAD KOLESNIKOV
GEORGIA TECH

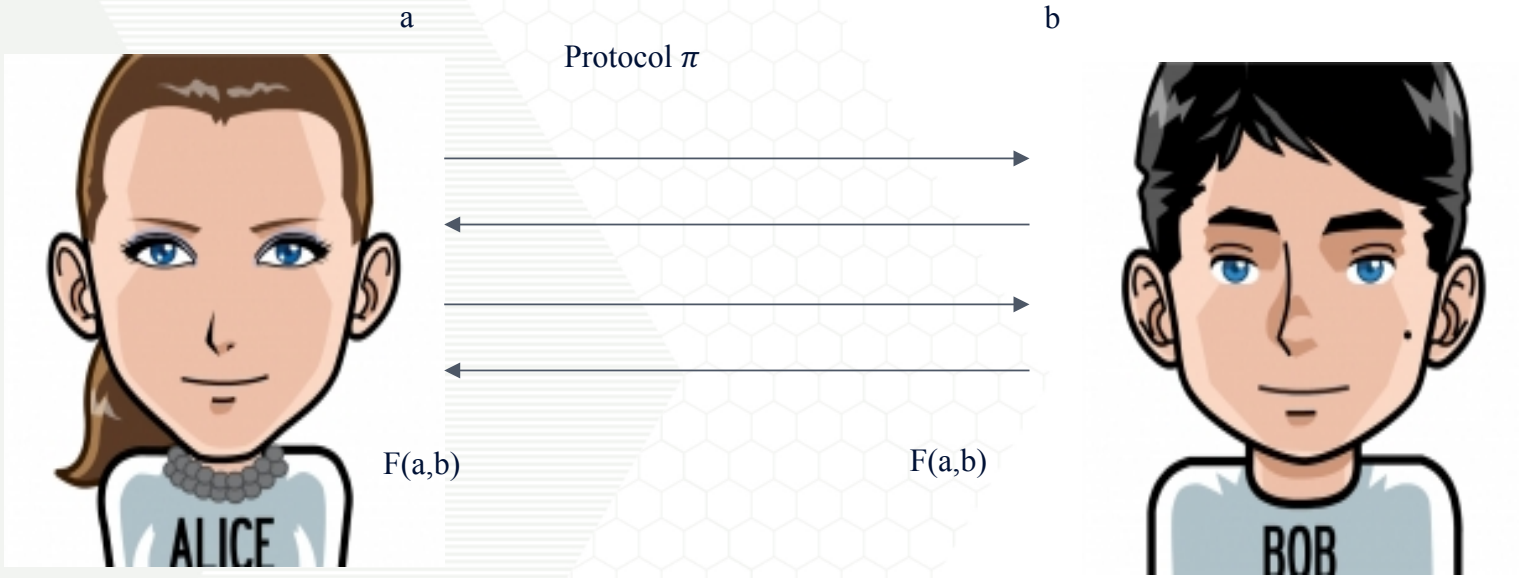
CREATING THE NEXT®

# HIGH-LEVEL OVERVIEW OF THE RESULT




In GC, if garbler knows the evaluated clause, don't need to generate/send inactive clauses

# DATA PRIVACY AND SECURE COMPUTATION

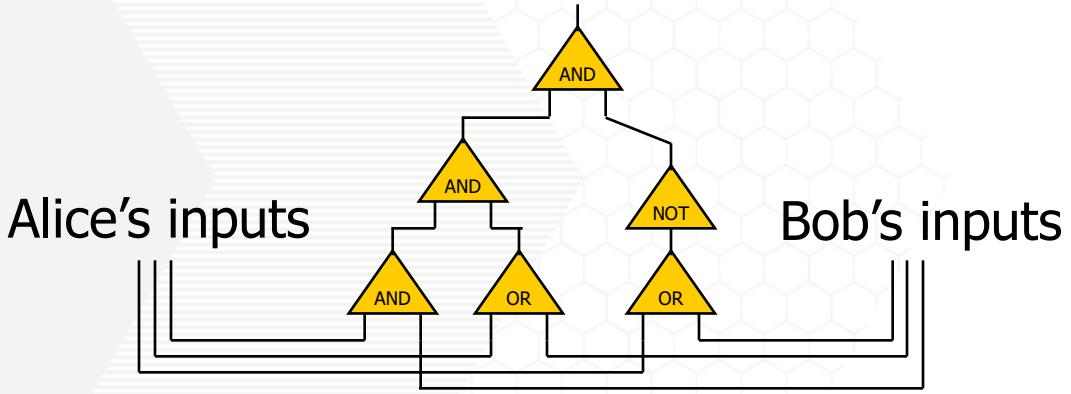


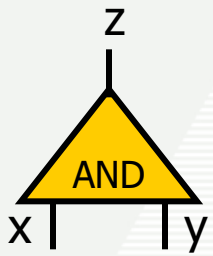

**Secure Function Evaluation (SFE)**

SFE How-to:
1) Given F, generate Boolean circuit C computing F
2) Securely evaluate C gate-by-gate

# GARBLED CIRCUIT

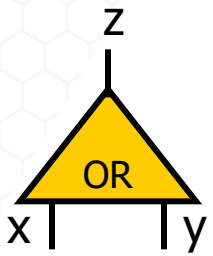
- Compute any function securely
- Represent the function as a boolean circuit




Truth table:

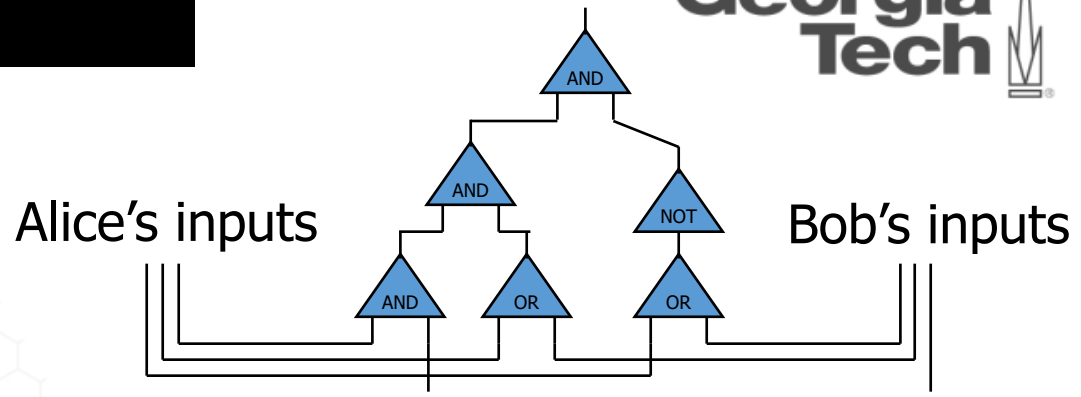
| x | y | z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




Truth table:

| x | y | z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# GARBLED CIRCUIT




**Overview:**

1. Alice prepares "garbled" version **C'** of C
2. Sends "encrypted" form **x'** of her input x
3. Allows bob to obtain "encrypted" form **y'** of his input y
4. **Bob can compute from C',x',y' the "encryption" z' of z=C(x,y)**

   **Think "Evaluation under encryption"**
5. Bob sends **z'** to Alice and she decrypts and reveals to him z

**Crucial properties**:

1. Bob never sees Alice's input x in unencrypted form.
2. Bob can obtain encryption of y without Alice learning y.
3. Neither party learns intermediate values.
4. Remains secure even if parties try to cheat.

# BOOLEAN CIRCUITS

- Circuit representation is inefficient
  - Random access (lots of work)
    - ORAM
  - Duplicate `if/switch` clauses
    - Universal circuit
    - [KKW17] – circuit embeddings
    - today

# CIRCUITS WITH BRANCHES

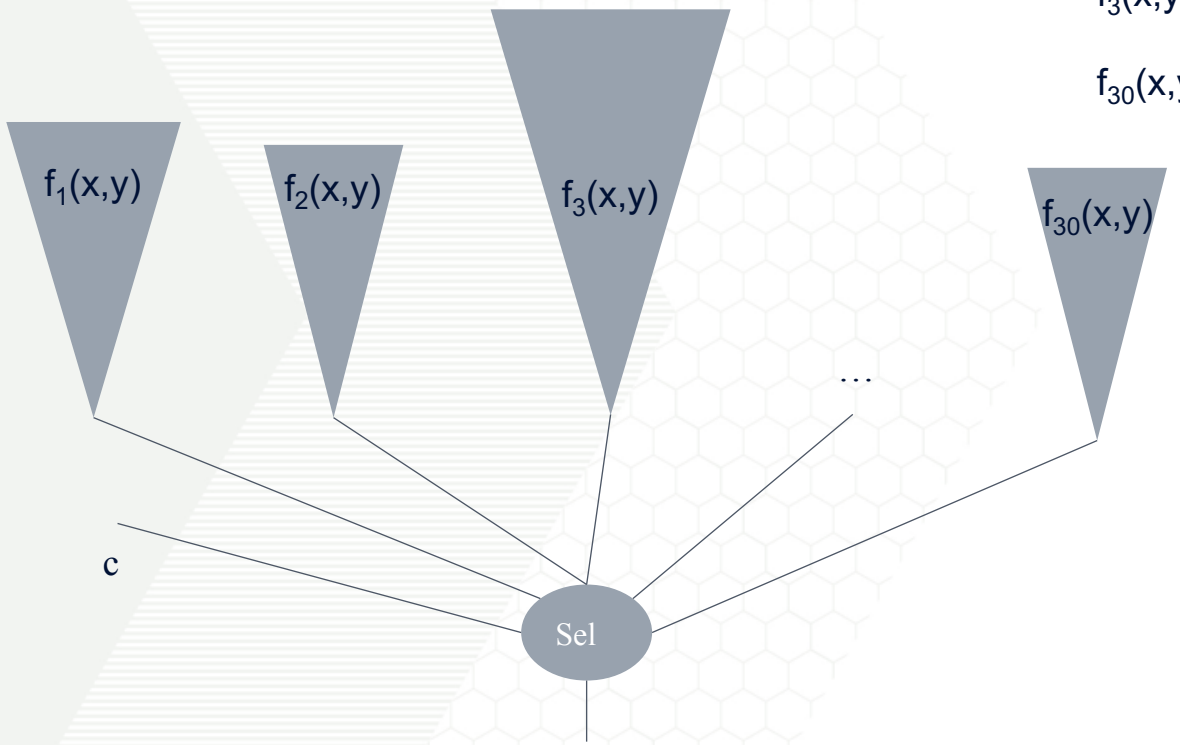
- $F(c,x,y) = f_c(x,y)$, where

$f_1(x,y) = y/2<x<y$
$f_2(x,y) = xy > 9000$
$f_3(x,y) = Ham(x,y) < 30$
…
$f_{30}(x,y) = x^2+y^2<9000$

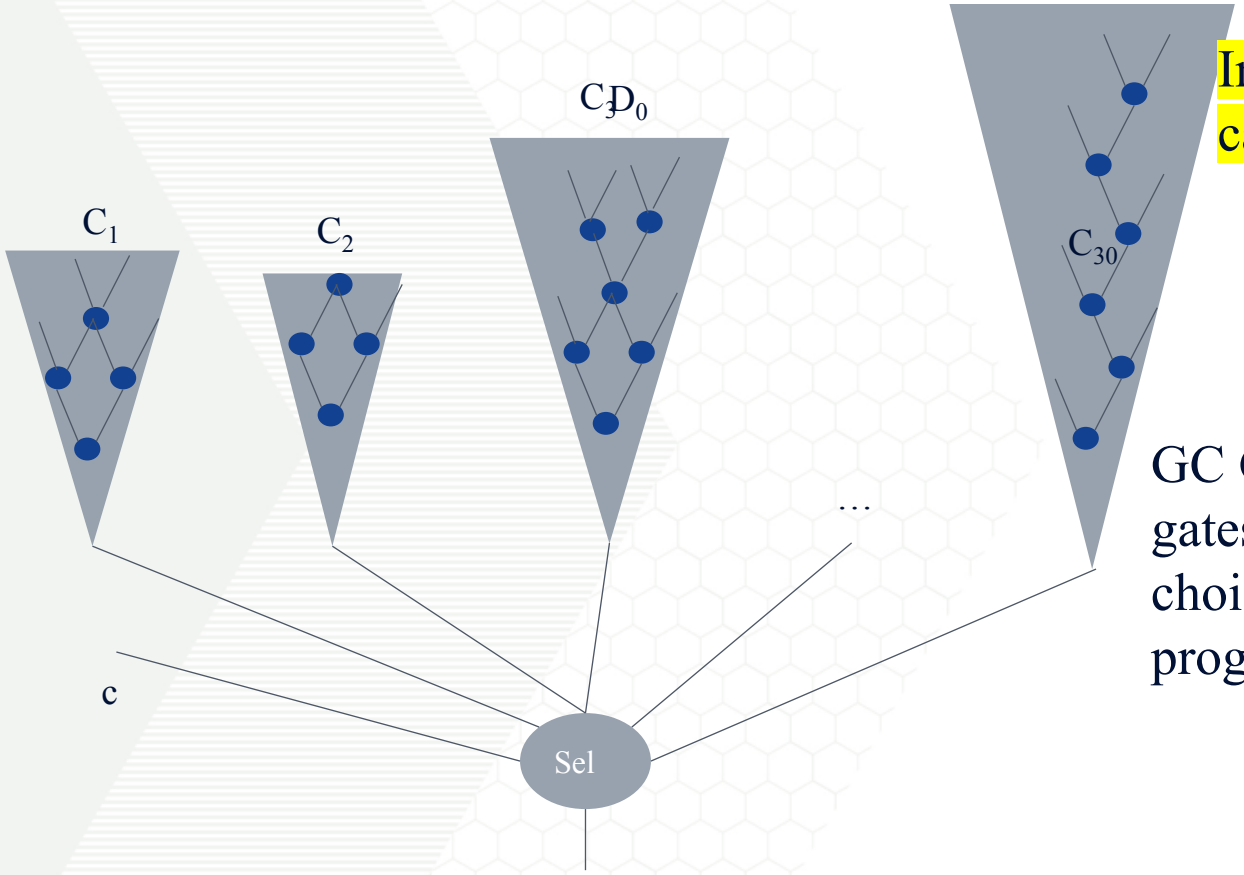
# CIRCUITS WITH BRANCHES

- $F(c,x,y) = f_c(y)$, where

$f_1(x,y)$ = $y/2<x<y$
$f_2(x,y)$ = $xy > 9000$
$f_3(x,y)$ = $Ham(x,y) < 30$
…
$f_{30}(x,y) = x^2+y^2<9000$



# EVALUATION VIA CIRCUIT EMBEDDING



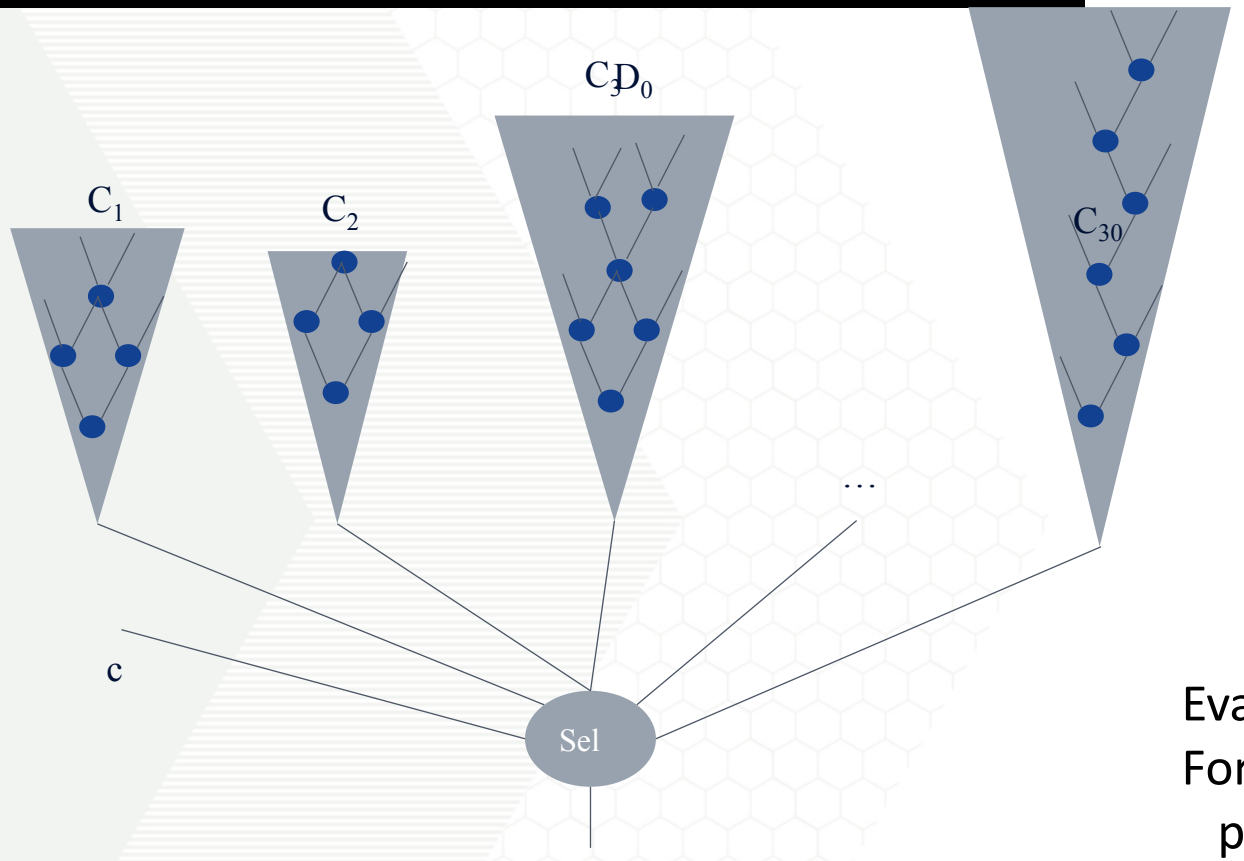

In GC gate function is hidden and can be set by Generator to anything.

GC Generator will program the gates according to its input choice c. Any of $C_1 \ldots C_{30}$ is programmable in $D_0$

# EMBED HOW?

- Previous work:
  - Theorem: finding optimal embedding is NP-hard
  - Naïve evaluation of all $C_1 \ldots C_k$ works
  - Universal circuit works
    - Too expensive for smaller switches ($\approx 5\, n \log n$, where $n$ is max circuit size)
  - Hand-designs [PSS09]
    - Doable only for trivial circuit combinations

# KEY: VIEW GC AS A STRING PARSED AS A COLLECTION OF GARBLED TABLES




If Garbler knows which target branch $C_t$ is to be evaluated:

- Garble and send $GC = GC_t$ and input labels $X$
- To Evaluator, $GC_t \approx GC_i$ (if pad the size)

Constr. 1:

Evaluator (on message $GC$):
For i:=1 to 30
  parse $GC$ as $GC_i$
  set $F_i = (T_i, GC)$
  compute $Y_i = Eval\ (F_i, X)$

Postprocess $Y'_t = SelRecode\ (t, Y_i)$

# SO WHAT DID WE GET

Only works if Gen knows the evaluated branch

| | Now | Before |
|---|---|---|
| Work (Gen) | $\approx O\left(\max_i \lvert C_i \rvert\right)$ | $\approx O(\sum \lvert C_i \rvert)$ |
| Work (Eval) | $\approx O(\sum \lvert C_i \rvert)$ | $\approx O(\sum \lvert C_i \rvert)$ |
| Comm | $\approx O\left(\max_i \lvert C_i \rvert\right)$ | $\approx O(\sum \lvert C_i \rvert)$ |

# PERFORMANCE

Only works if Gen knows the evaluated branch

| Num branches | Performance improvement |
|---|---|
| $5$ | $\approx 5\times$ |
| $20$ | $\approx 20\times$ |
| $100$ | $\approx 100\times$ |

Need one extra round of comm to run ${Y'}_t = SelRecode\ (t, Y_i)$. Its cost is independent of circuit size and is concretely small

# MOTIVATION FOR PRIVATE FUNCTION SFE (PF-SFE)

Private DB:
-Policy allows for query types $T_1, .., T_{20}$, Client wants to hide which query type is being run

CPU emulation:
- CPU evaluates instructions one by one, implemented via SFE. We want to hide the program being run. [WGMK17] implemented MIPS CPU using 36 different instructions (and each step generates and sends 36 GCs, only one of which is used).

# PRESENTATION IN THE BHR FRAMEWORK

We slightly depart from the standard GC syntax and semantics.
Goal: reuse all accumulated (and future) body of work in BHR terminology.

Result: we extend Bellare-Hoang-Rogaway framework to accommodate the change.

Main difference: Separation of circuit topology T from cryptographic material E.

# PRESENTATION IN THE BHR FRAMEWORK

1. Let F be a BHR GC. We syntactically separate topology T from cryptographic material E (garbled tables). We write F = (T;E), thus enabling consideration of a GC (T';E).
2. Adjust the definitions to support evaluation under a "wrong" function encoding, and further, to require that Eval will not detect whether it operates with a "right" or "wrong" encoding.

    2a. A BHR circuit garbling scheme is *Topology-decoupling circuit garbling* scheme if above holds.
3. Some additional low-level adjustments (e.g. handling bitwise output decoding).

# REUSING BHR MACHINERY

Main point:
We restrict BHR. The only generalization is the F = (T;E) parsing, which was not exercised before. => all BHR theorems apply.

Our notion is a special case of BHR garbling scheme, and thus we can keep the BHR function definitions and correctness and security requirements as is. This is because we restricted the syntax of the BHR notions. Our only generalization (allowing to evaluate under different topology), is not exercised in BHR definitions. Therefore, all BHR notation and definitions retain their meaning and are reused.

# REUSING BHR MACHINERY

Theorems (informal):

Theorem 1: Construction 1 is a secure SFE protocol in the semi-honest model, if the employed garbling scheme is topology-decoupling circuit garbling.

Theorem 2: Half-gates scheme is topology-decoupling circuit garbling.
Theorem 3: Free-XOR scheme is topology-decoupling circuit garbling.

To use any BHR scheme with our approach:

1. Prove (amend if necessary) that scheme is topology –decoupling.