Protecting Privacy in Computation

UF UNIVERSITY of FLORIDA

Space Vehicles Directorate
Technology for the Final Frontier

Duke UNIVERSITY

AFRL Sensors Directorate

TEXAS
The University of Texas at Austin

Munitions Directorate

UC SANTA CRUZ

Motivation: Conjunction Analysis

- Many objects exist in space
- Many of these are satellites… some of which are undisclosed
- Question: How how prevent collisions without revealing what's in the air?
- More generally: how to ensure data communicated by autonomous agents stays secure and private?

Privacy-Preserving Cryptography

- Homomorphic Encryption

  $$E(m_1) \star E(m_2) = E(m1 \star m2) \forall m_1, m_2 \in M$$

  - Fully homomorphic (FHE) scheme supports addition and multiplication as operations
  - Popularized by Gentry's 2009 breakthrough using ideal lattices

    $$L = \sum_{i=1}^{n} \vec{b_i} * v_i, v_i \in \mathbb{Z}$$

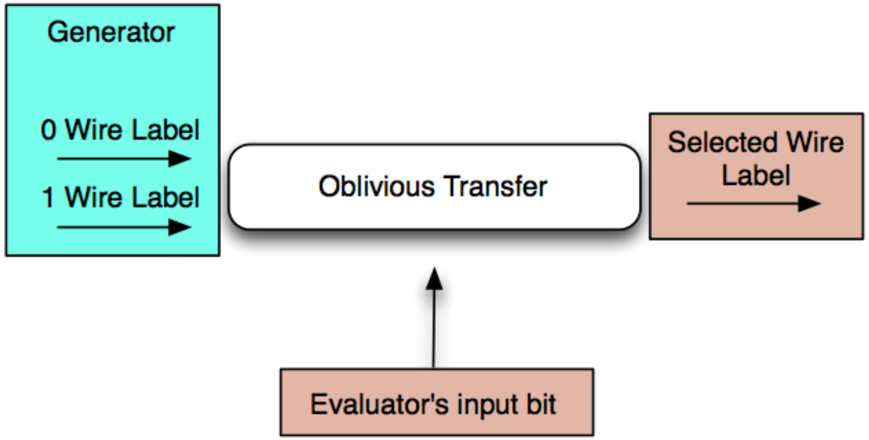
  - Downside: computationally infeasible for many years (around 10^12 initially for ideal lattices)
  - Performance increased but still not great for near-realtime

Secure Multiparty Computation

- Allow joint computation of a function without revealing input from either party
- Cryptographically secured through the use of *garbled* Boolean circuits and *oblivious transfer* of data from circuit generator to evaluator

Garbled Gate
Lock($A_0 \| \| B_0$, O_0)
Lock($A_0 \| \| B_1$, O_0)
Lock($A_1 \| \| B_0$, O_0)
Lock($A_1 \| \| B_1$, O_1)

lock Output O under inputs A and B



Garbling and Evaluation

Let $f : \{0,1\}^A \times \{0,1\}^B \rightarrow \{0,1\}^j \times \{0,1\}^k$ be a computable function
- Receives input bits from 2 parties, produces output bits for each party

Garble circuit with block cipher $\langle E, G \rangle$, then compute $(k_0, k_1) \leftarrow (G(1^n), G(1^n))$, which represent logical 0 and 1 values.

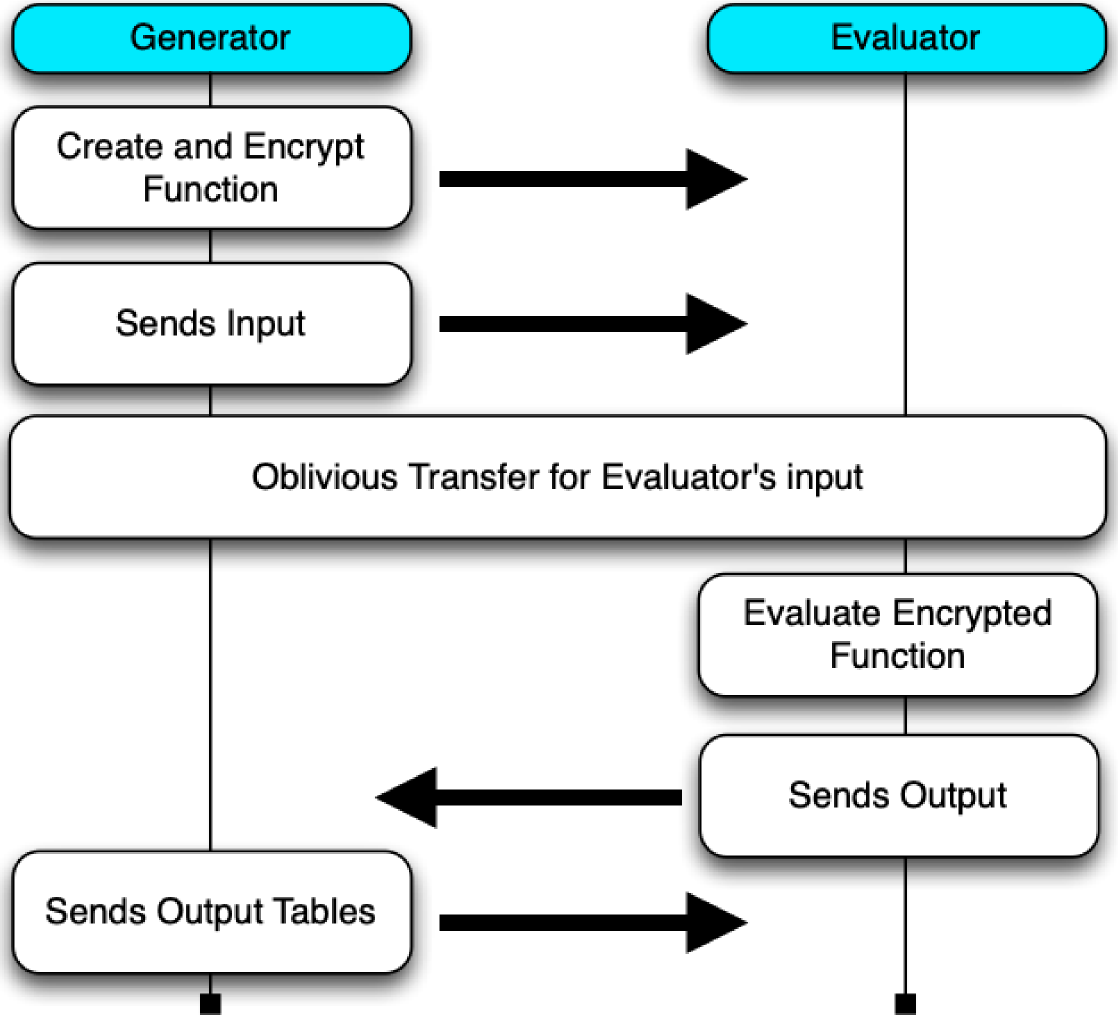
For each gate, if the truth table is $[v_{0,0}, v_{0,1}, v_{1,0}, v_{1,1}]$, the generator computes the following ciphertext:

$$\begin{bmatrix} E_{k_{l,0}}(E_{k_{r,0}}(k_{v_{0,0}})), E_{k_{l,0}}(E_{k_{r,1}}(k_{v_{0,1}})) \\ E_{k_{l,1}}(E_{k_{r,0}}(k_{v_{1,0}})), E_{k_{l,1}}(E_{k_{r,1}}(k_{v_{1,1}})) \end{bmatrix}$$

Circuit Evaluation

- Generator sends evaluator the input wire keys
- 1-of-2 oblivious transfer for each input wire
  $k_0 = (v - x_0)^d \bmod N, k_1 = (v - x_1)^d \bmod N$
- Evaluator decrypts output gates $E_{k_{r,*}}(E_{k_{l,*}}(k_{v_{bit_l,bit_r}}))$
  - $k_{l,*}$ and $k_{r,*}$ are keys the evaluator has
  - $k_{v_{bit_l,bit_r}}$ is the garbled truth table entry selected by the point and permute bits bit_l and bit_r

Semi-Honest Protocol



Generator

Evaluator

Create and Encrypt Function

Sends Input

Oblivious Transfer for Evaluator's input

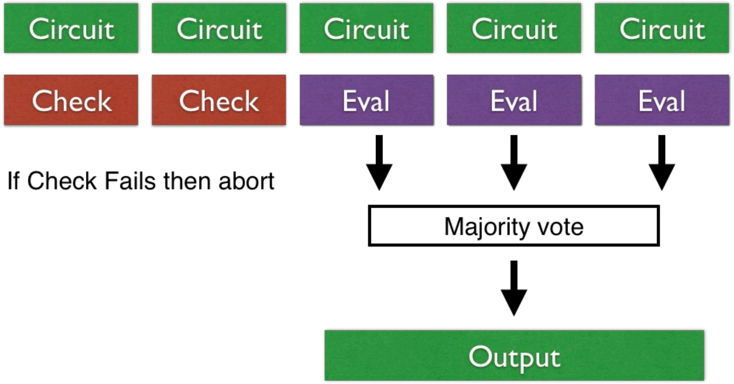
Evaluate Encrypted Function

Sends Output

Sends Output Tables

Malicious Security Model

- In presence of active adversaries, data can be
  - Maliciously generated
  - Selective failure on input
  - Inconsistent on input or output
- Solution: Perform computation *N* times to prevent use of incorrect circuit
  - Open *S* of *N* circuits (cut and choose)



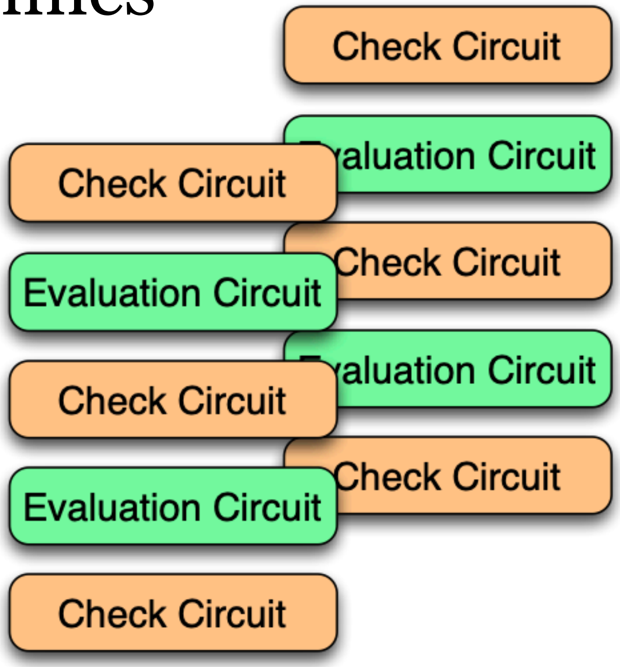
Circuit Circuit Circuit Circuit Circuit

Check Check Eval Eval Eval

If Check Fails then abort

Majority vote

Output



Check Circuit

valuation Circuit

Check Circuit

Check Circuit

Evaluation Circuit

valuation Circuit

Check Circuit

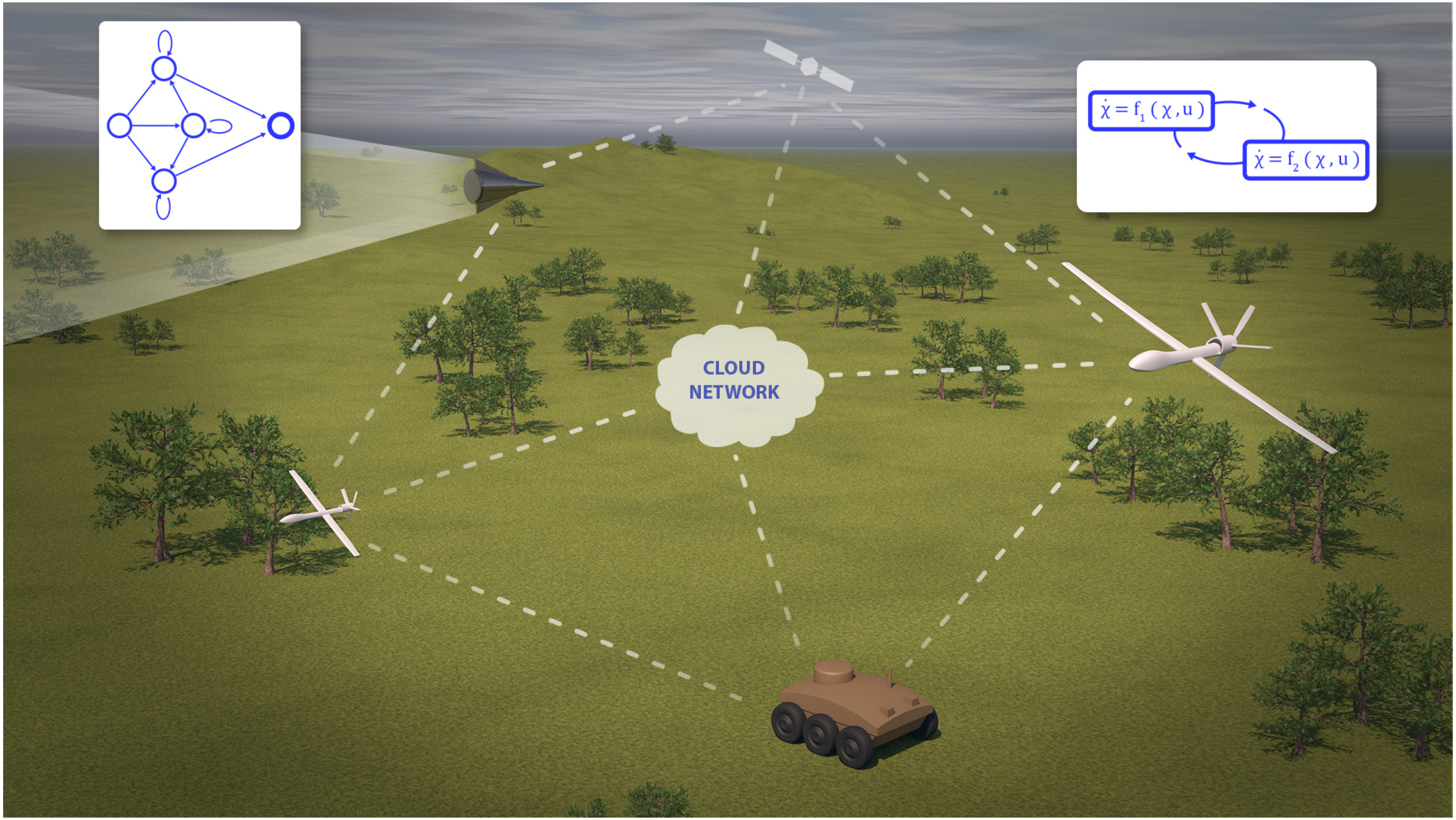
Check Circuit

Evaluation Circuit

Check Circuit

Outsourcing Evaluation

- Setting: Resource-constrained autonomous agent (Alice) communicating with better provisioned service (Bob). Alice also has access to a third-party compute service (Cloud).
- Goal: Alice and Bob securely compute a two-party function using garbled circuits. We consider the case where Bob generates the circuit and Alice evaluates.
- Security:
  - Preserve input and output privacy from both the other party and the cloud
  - Security in the malicious setting



UF UNIVERSITY of FLORIDA

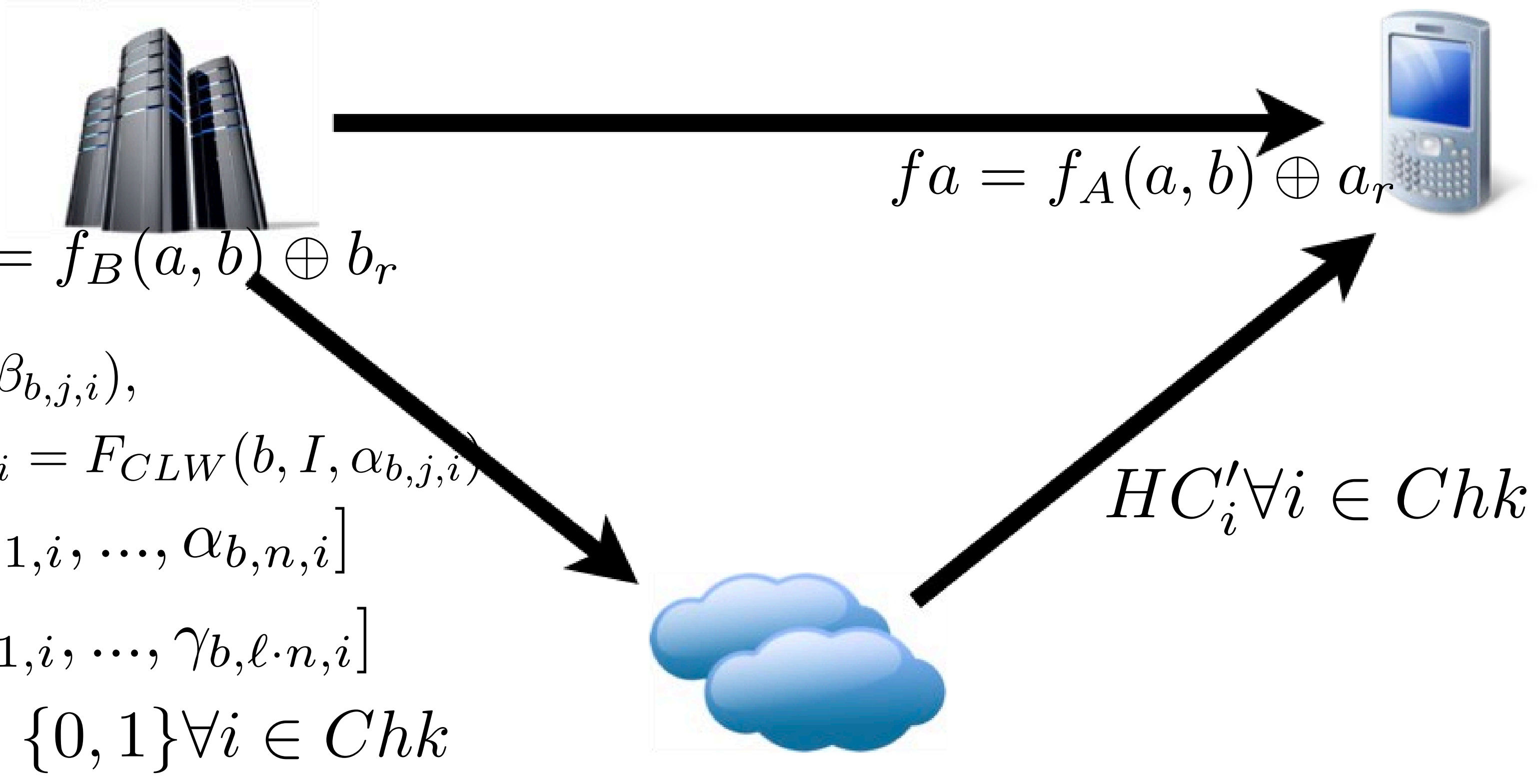
Duke UNIVERSITY

TEXAS The University of Texas at Austin

UC SANTA CRUZ

Protocol Stages

I: Circuit generation & check



Bob
(generator)

Alice
(evaluator)

$fa = f_A(a,b) \oplus a_r$

$fb = f_B(a,b) \oplus b_r$

$H_2(\beta_{b,j,i}),$

$\beta_{b,j,i} = F_{CLW}(b, I, \alpha_{b,j,i})$

$[\alpha_{b,1,i}, \ldots, \alpha_{b,n,i}]$

$[\gamma_{b,1,i}, \ldots, \gamma_{b,\ell\cdot n,i}]$

$b = \{0,1\} \forall i \in Chk$

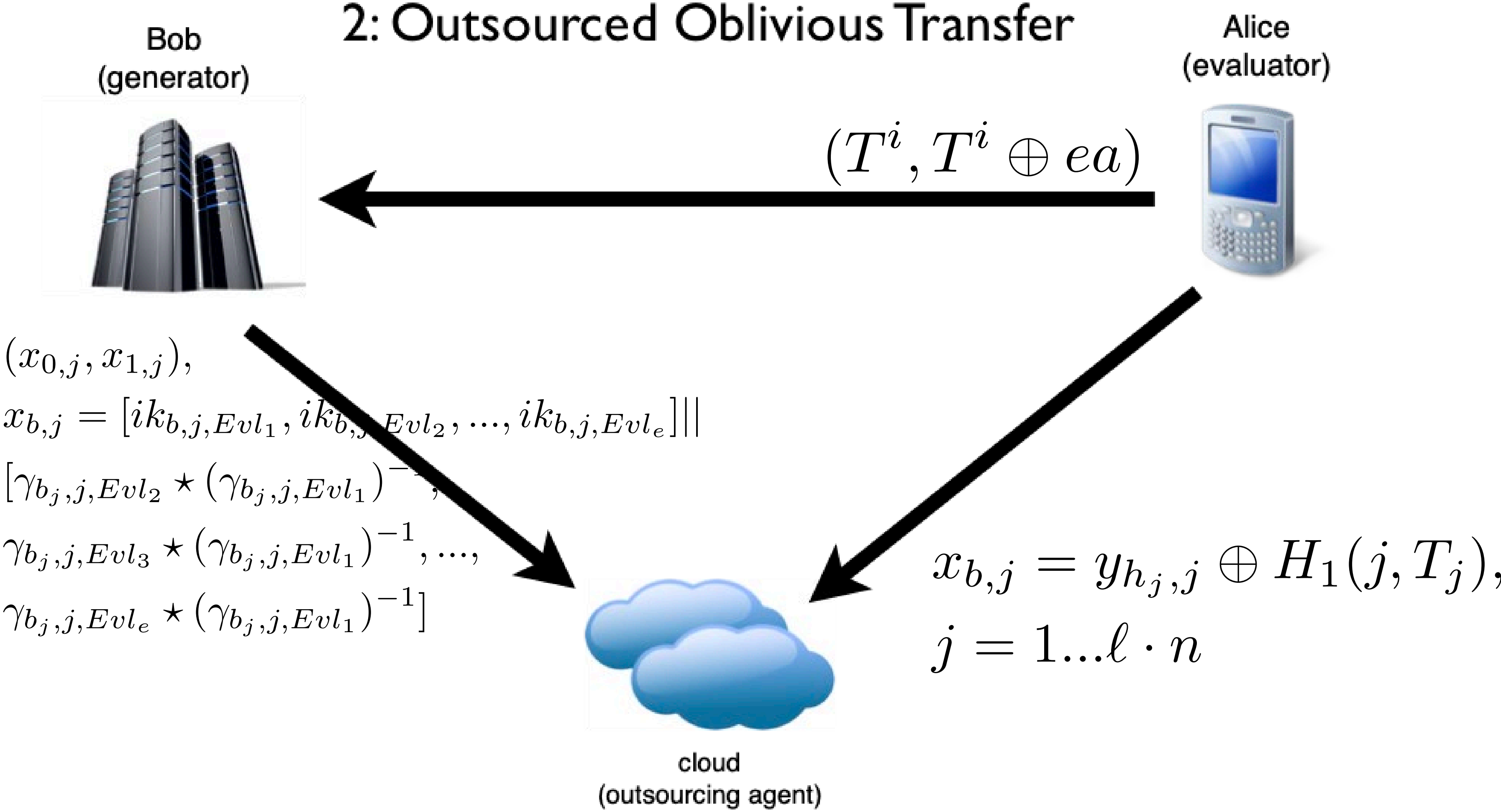
$HC'_i \forall i \in Chk$

cloud
(outsourcing agent)

$Garble(C, rc_i) = GC'_i \forall i \in Chk$

Protocol Stages

2: Outsourced Oblivious Transfer



Bob
(generator)

Alice
(evaluator)

$(T^i, T^i \oplus ea)$

$(x_{0,j}, x_{1,j}),$

$x_{b,j} = [ik_{b,j,Evl_1}, ik_{b,j,Evl_2}, \ldots, ik_{b,j,Evl_e}]||$

$[\gamma_{b_j,j,Evl_2} \star (\gamma_{b_j,j,Evl_1})^{-1},$

$\gamma_{b_j,j,Evl_3} \star (\gamma_{b_j,j,Evl_1})^{-1}, \ldots,$

$\gamma_{b_j,j,Evl_e} \star (\gamma_{b_j,j,Evl_1})^{-1}]$

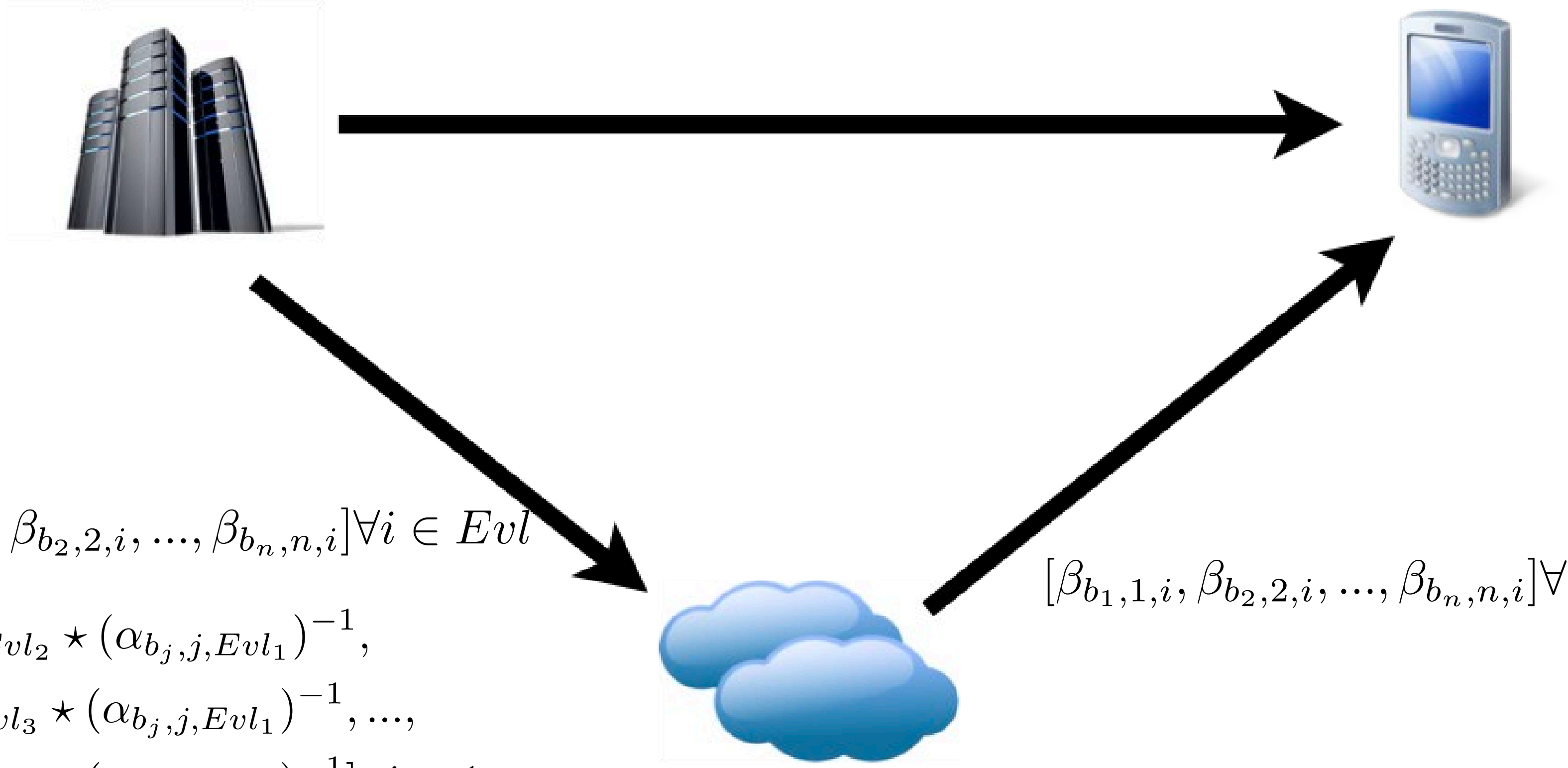
$x_{b,j} = y_{h_j,j} \oplus H_1(j, T_j),$

$j = 1 \ldots \ell \cdot n$

cloud
(outsourcing agent)

Protocol Stages

3: Input consistency check



Bob
(generator)

Alice
(evaluator)

$$[\beta_{b_1,1,i}, \beta_{b_2,2,i}, \ldots, \beta_{b_n,n,i}] \forall i \in Evl$$

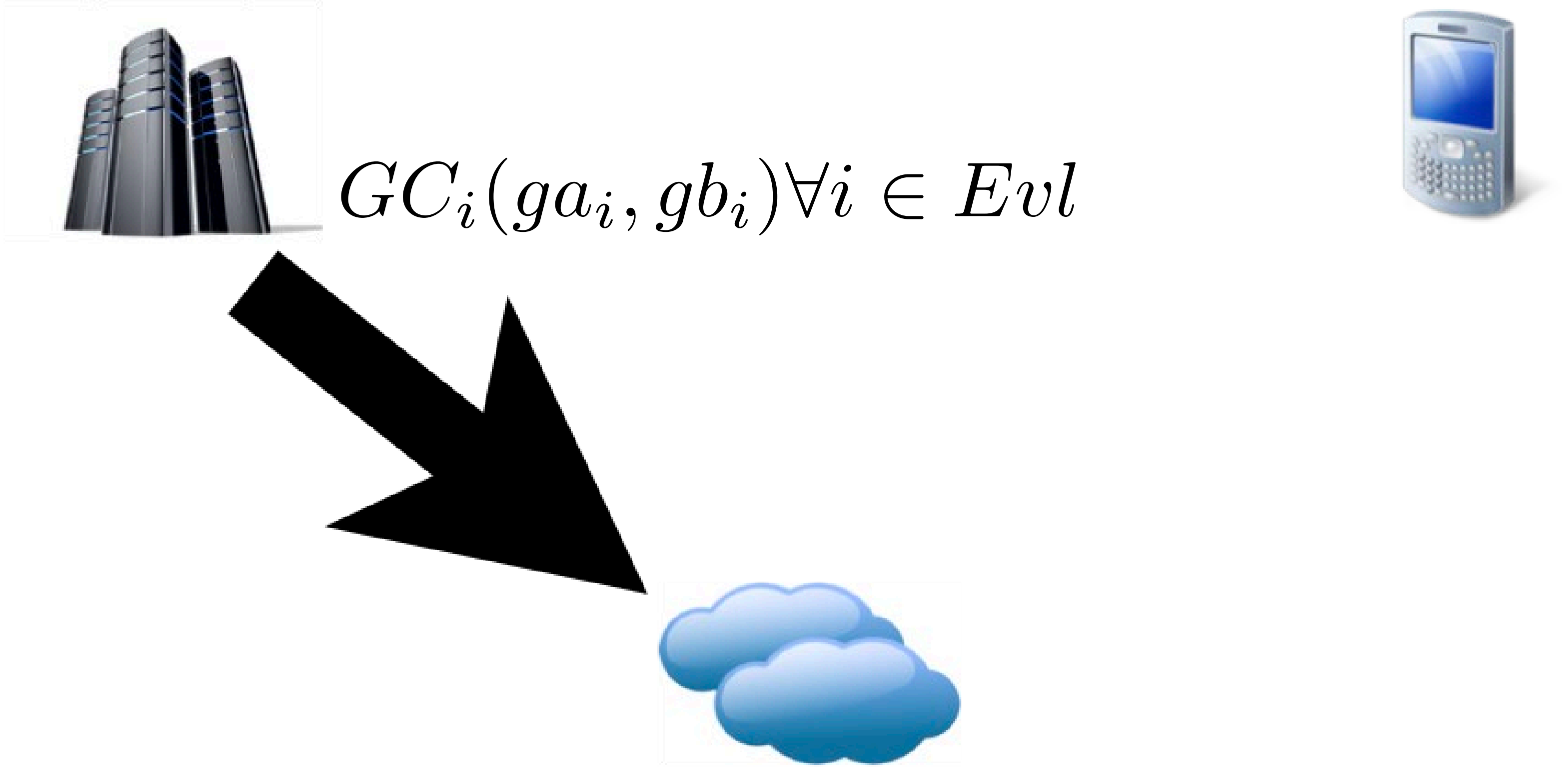
$$[\alpha_{b_j,j,Evl_2} \star (\alpha_{b_j,j,Evl_1})^{-1},$$
$$\alpha_{b_j,j,Evl_3} \star (\alpha_{b_j,j,Evl_1})^{-1}, \ldots,$$
$$\alpha_{b_j,j,Evl_e} \star (\alpha_{b_j,j,Evl_1})^{-1}], j = 1..n$$

$$[\beta_{b_1,1,i}, \beta_{b_2,2,i}, \ldots, \beta_{b_n,n,i}] \forall i \in Evl$$

cloud
(outsourcing agent)

Protocol Stages

4: Evaluation



Bob
(generator)

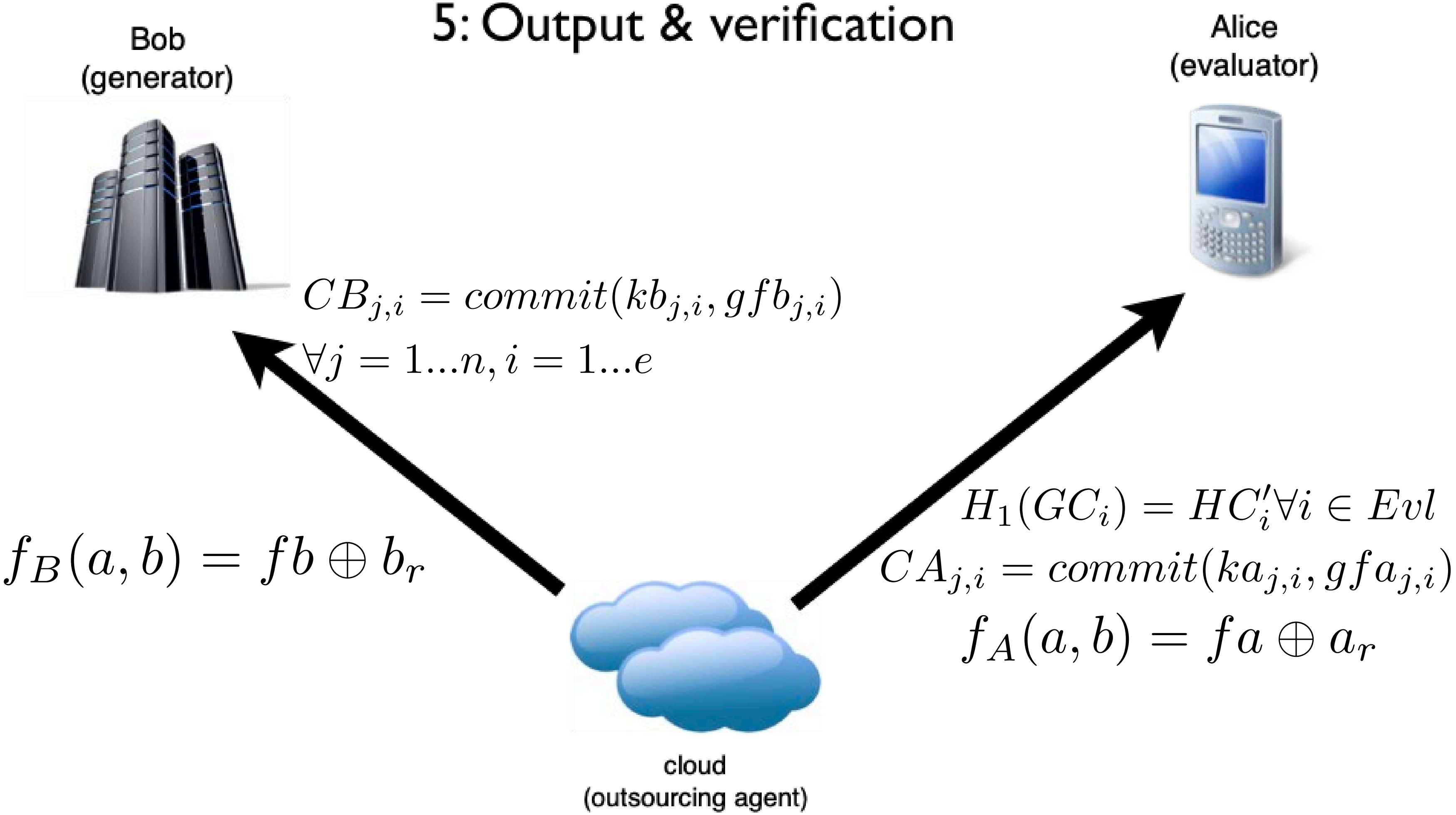
$$GC_i(ga_i, gb_i) \forall i \in Evl$$

Alice
(evaluator)

cloud
(outsourcing agent)

Protocol Stages

5: Output & verification



$$CB_{j,i} = commit(kb_{j,i}, gfb_{j,i})$$
$$\forall j = 1...n, i = 1...e$$

$$f_B(a,b) = fb \oplus b_r$$

$$H_1(GC_i) = HC'_i \forall i \in Evl$$
$$CA_{j,i} = commit(ka_{j,i}, gfa_{j,i})$$
$$f_A(a,b) = fa \oplus a_r$$

Security Analysis

- Build from Kreuter et al. and preserve security in
  - Garbled circuits
  - Input consistency between evaluation checks
  - Output integrity and majority check
  - Outsourced oblivious transfer

- Formal proofs of security in malicious model

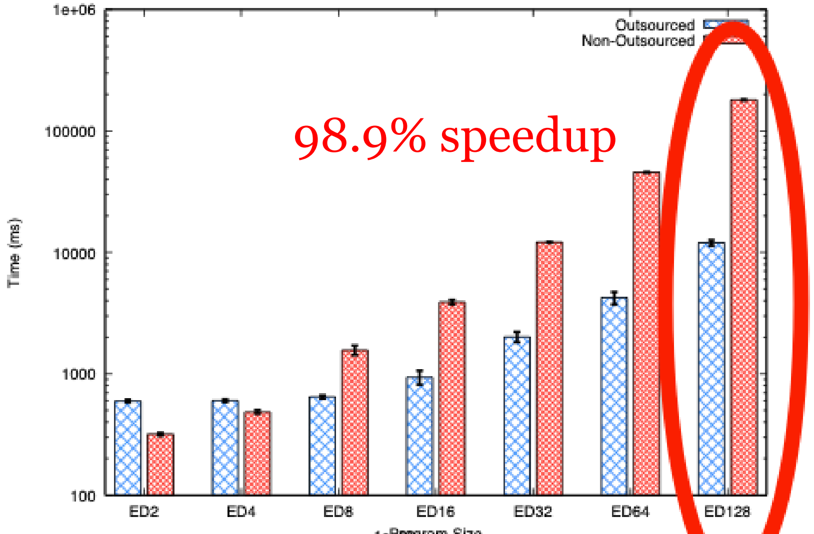
Definition 1 *A protocol securely computes a function f if there exists a set of probabilistic polynomial-time (PPT) simulators $\{Sim_i\}_{i\in[3]}$ such that for all PPT adversaries $(A_1, ..., A_3)$, x, z, and for all $i \in [3]$:*

$$\{REAL^{(i)}(k, x; r)\}_{k\in N} \overset{c}{\approx} \{IDEAL^{(i)}(k, x; r)\}_{k\in N}$$

Where $S = (S_1, ..., S_3)$, $S_i = Sim_i(A_i)$, and r is random and uniform.

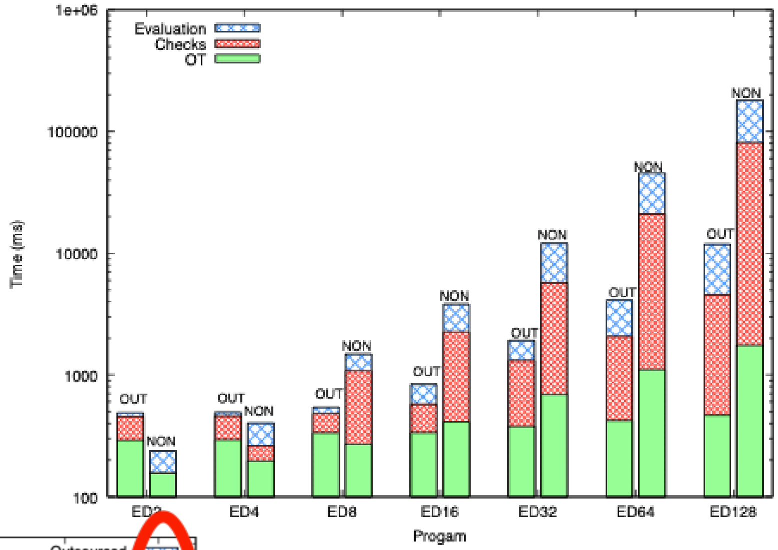
Edit Distance Performance Benchmarks

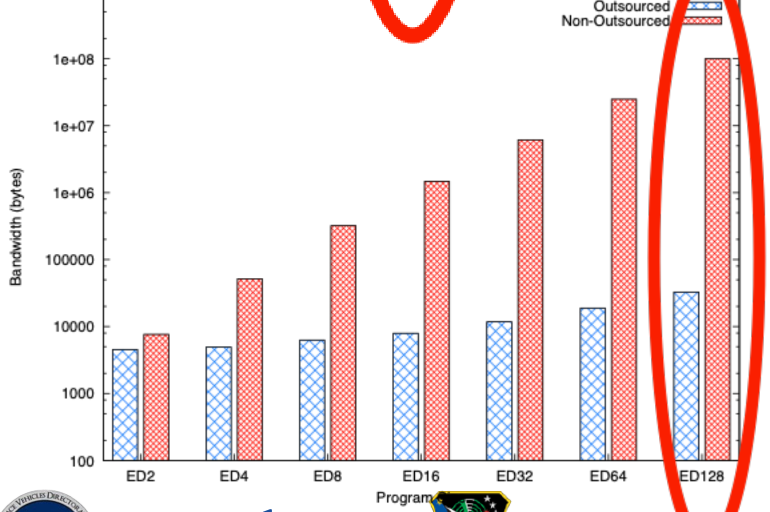
Total Runtime



98.9% speedup

Phase Runtime

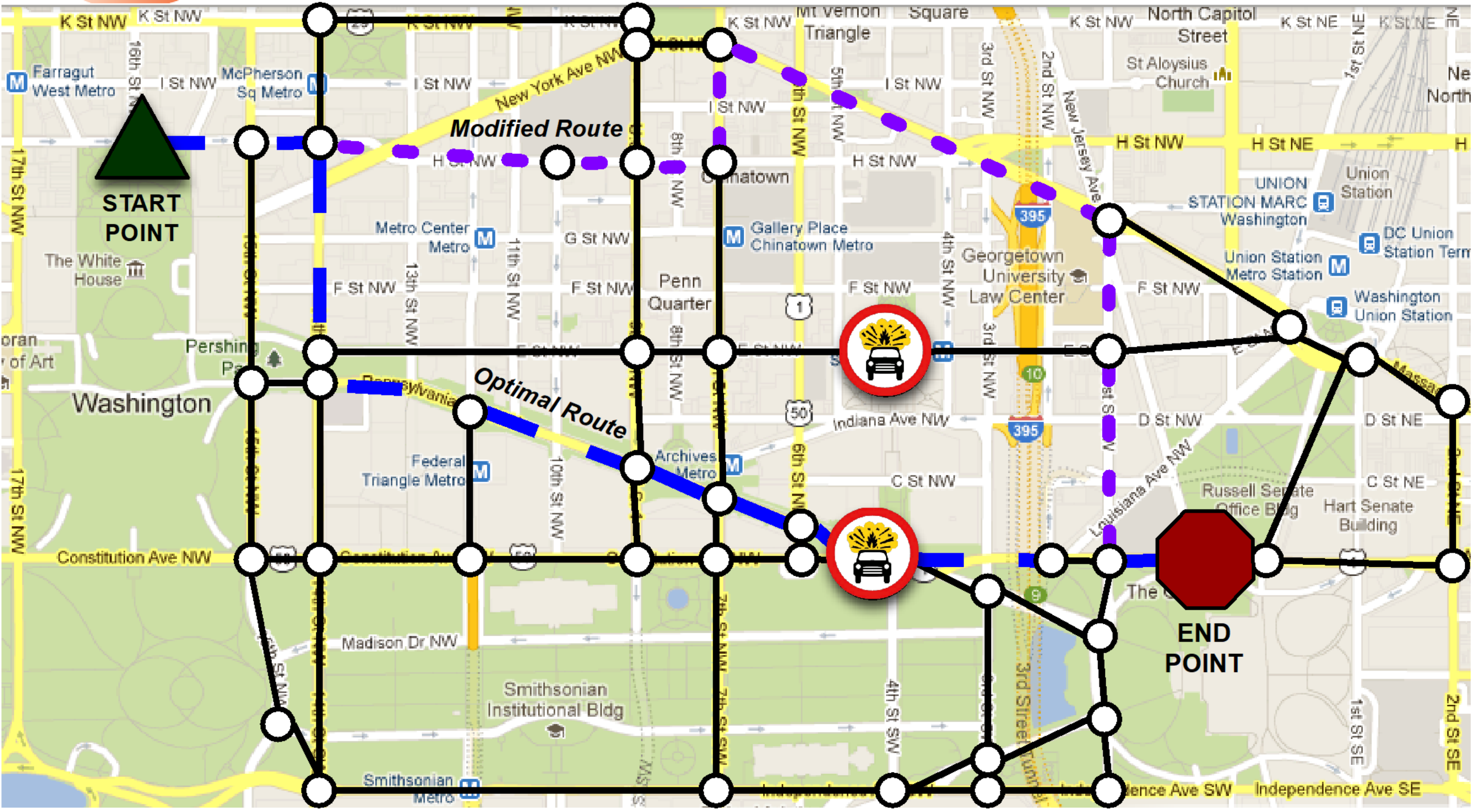




Total Bandwidth

3400X reduction in evaluator bandwidth

Privacy-Preserving Navigation



UF UNIVERSITY of FLORIDA

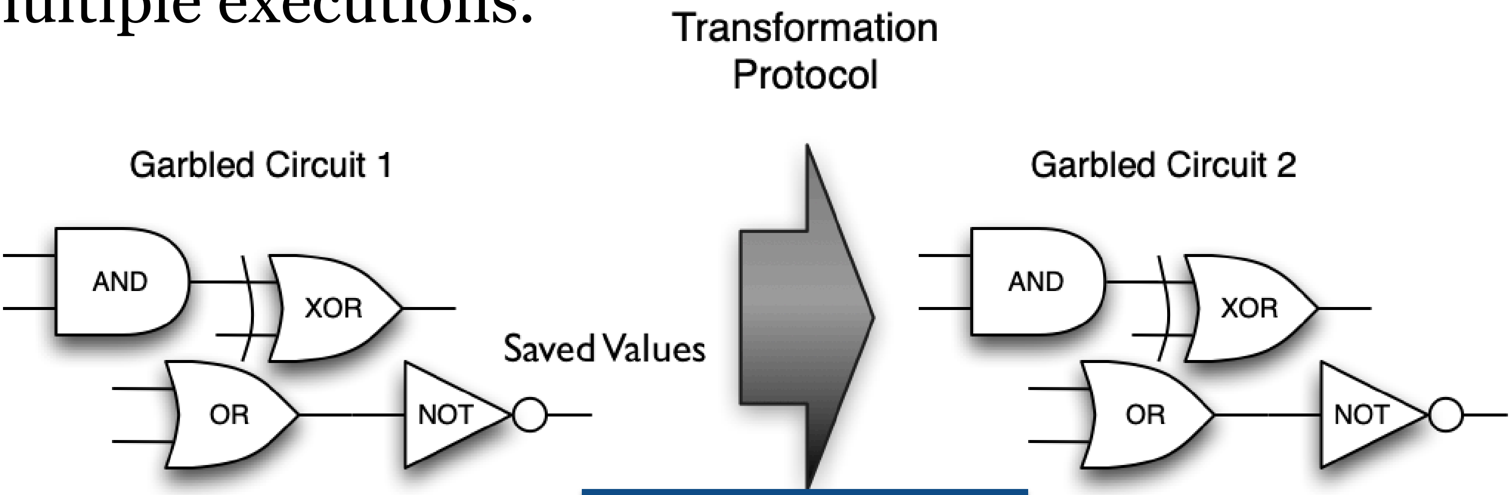
Duke UNIVERSITY

TEXAS The University of Texas at Austin

UC SANTA CRUZ

Optimization: Amortize Evaluation

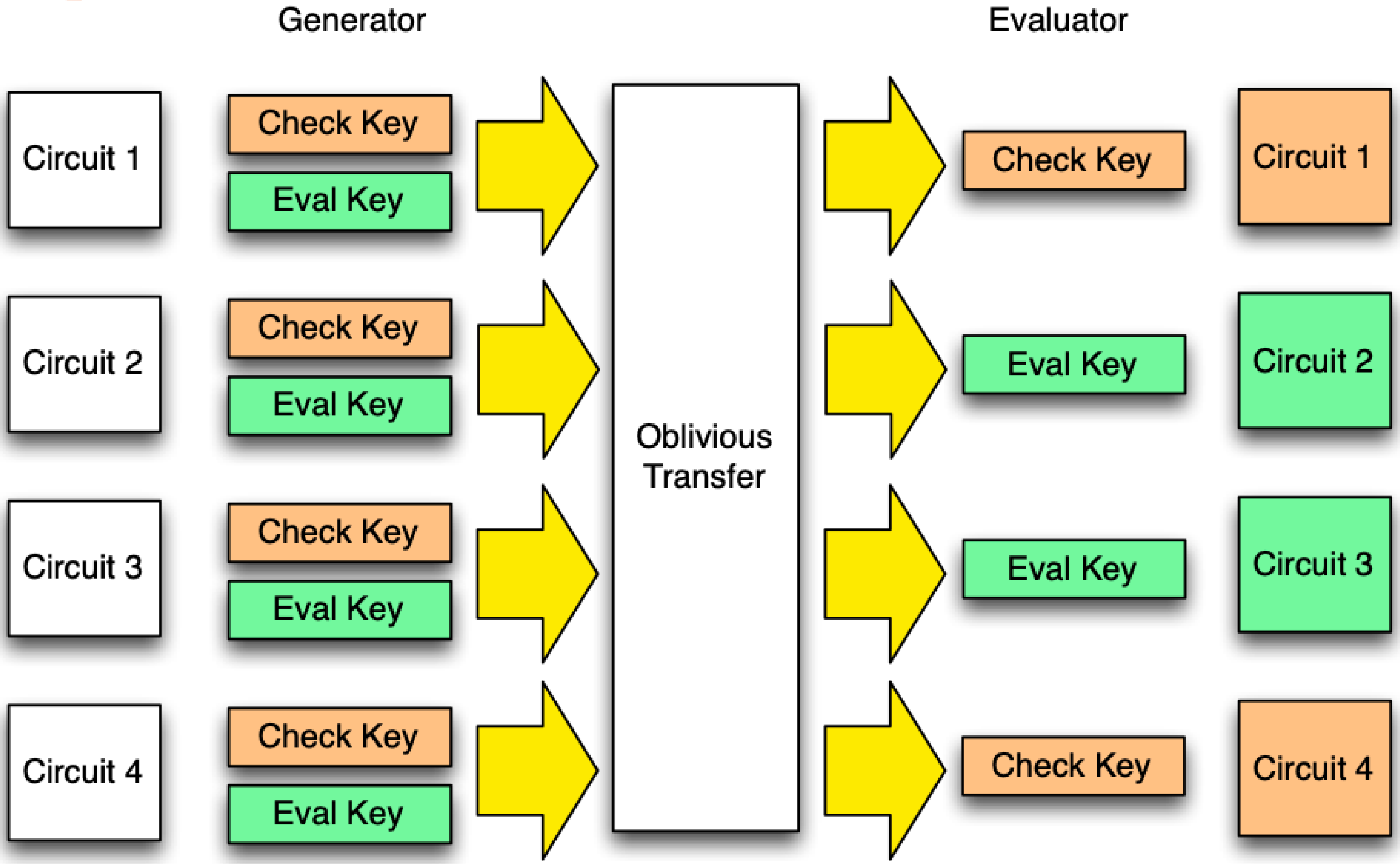
Rather than throw away state after every execution, reuse elements of circuits to amortize their cost across multiple executions.



Use 1 input garbled gates

$$TT_0 = Enc(Pre\_Out_{i,p} \oplus R) \oplus Post\_In_{i,p}$$
$$TT_1 = Enc(Pre\_Out_{i,1-p} \oplus R) \oplus Post\_In_{i,1-p}$$
$$p = permute$$
$$R = rand$$

Cut & Choose via OT



Generator

Evaluator

Circuit 1

Check Key

Eval Key

Circuit 2

Check Key

Eval Key

Circuit 3

Check Key

Eval Key

Circuit 4

Check Key

Eval Key

Oblivious Transfer

Check Key

Circuit 1

Eval Key

Circuit 2

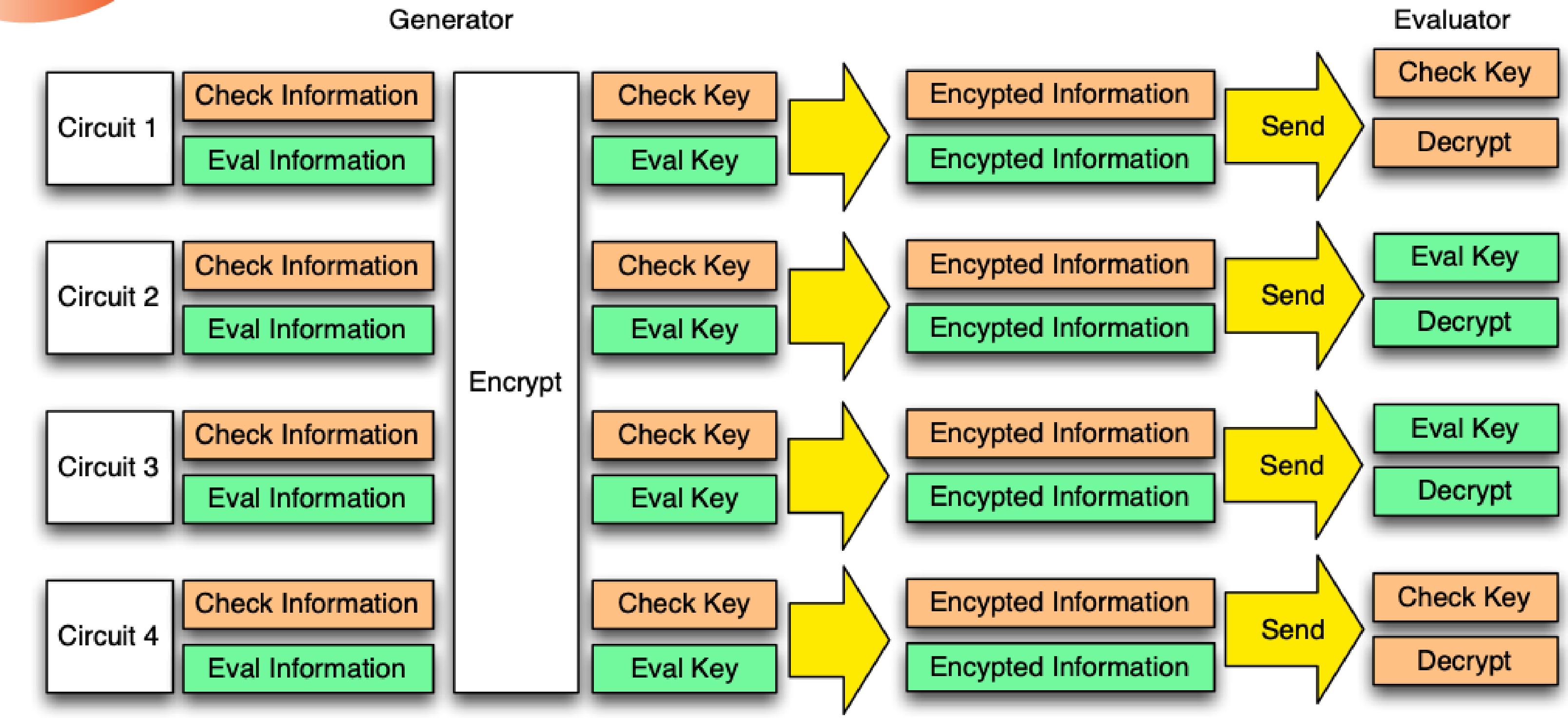
Eval Key

Circuit 3

Check Key

Circuit 4

Passing Circuit Information



Generator

Evaluator

Circuit 1: Check Information, Eval Information

Circuit 2: Check Information, Eval Information

Circuit 3: Check Information, Eval Information

Circuit 4: Check Information, Eval Information

Encrypt

Check Key, Eval Key → Encypted Information, Encypted Information → Send → Check Key, Decrypt

Check Key, Eval Key → Encypted Information, Encypted Information → Send → Eval Key, Decrypt

Check Key, Eval Key → Encypted Information, Encypted Information → Send → Eval Key, Decrypt

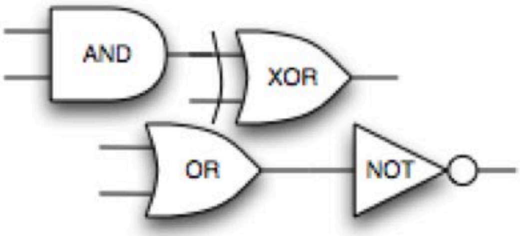
Check Key, Eval Key → Encypted Information, Encypted Information → Send → Check Key, Decrypt

Algorithm 0: PartialComputation

```
Input  : Circuit_File, Bit_Security, Number_of_Circuits, Inputs, Is_First_Execution
Output: Circuit File Output
Cut_and_Choose(is_First_Execution)
Eval_Garbled_Input ← Evaluator_Input(Eval_Select_Bits, Possible_Eval_Input)
Generator_Input_Check(Gen_Input)
Partial_Garbled_Input ← Partial_Input(Partial_Output$_{time-1}$)
Garbled_Output, Partial_Output ← Circuit_Execution(Garbled_Input (Gen, Eval,
  Partial))
Circuit_Output(Garbled_Output)
Partial_Output(Partial_Output)
```

Operation

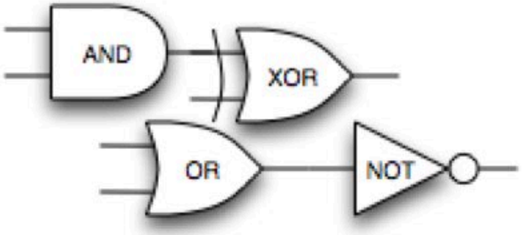
Garbled Circuit 1



- Get Keys Via OT
- Use keys for transfer of circuit information
- GC computation
- Save keys from OT
- Save wire values

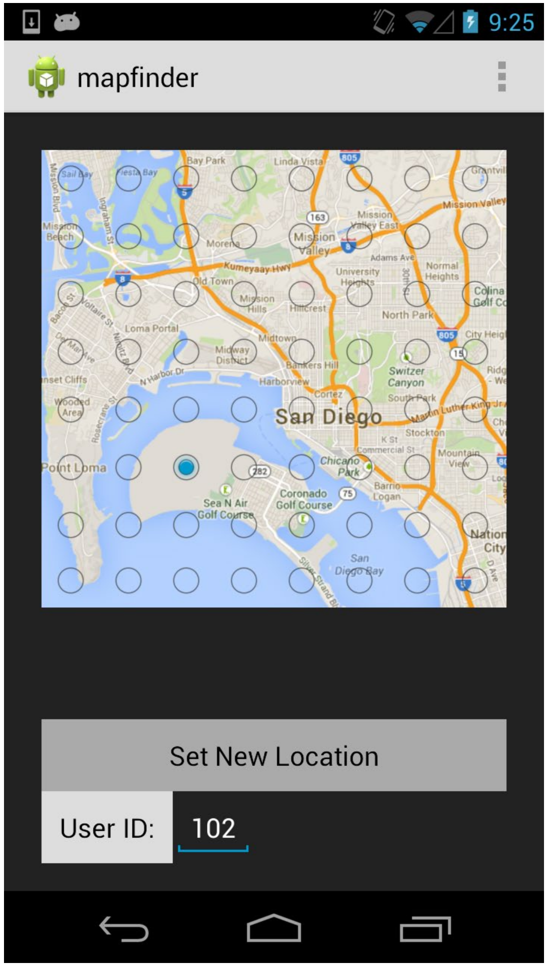
Saved Values →

Garbled Circuit 2



- Load keys from GC1 OT
- Use keys for transfer of circuit information
- Load wire values
- Create and evaluate transformation gates
- GC computation
- Save wire values

Performance with Amortization



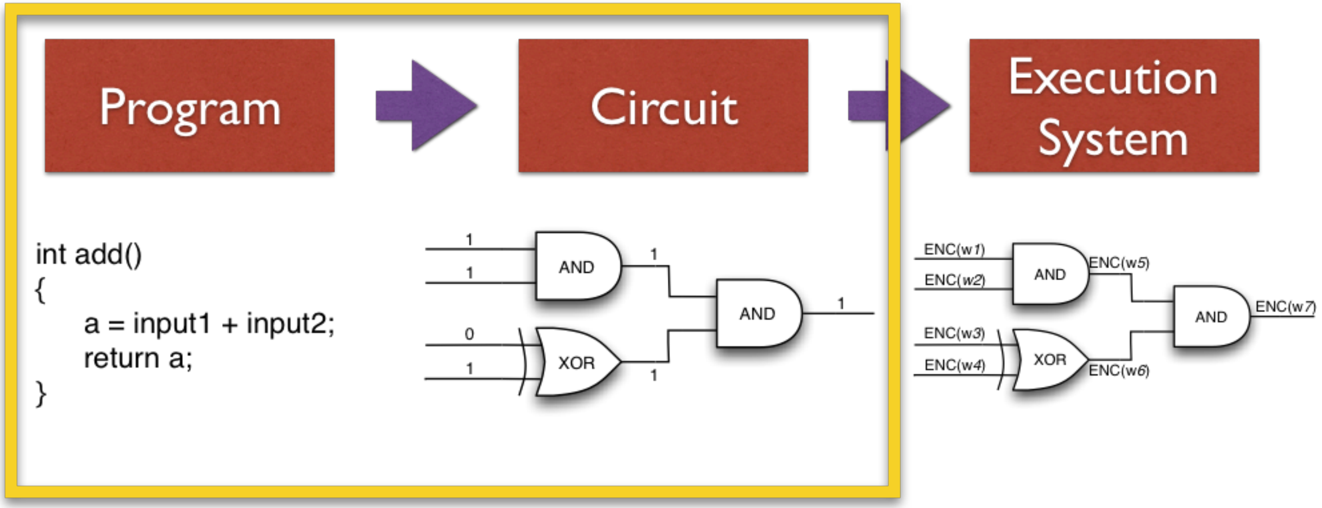
User ID: 102

	64 Circuits			256 Circuits		
	CMTB	PartialGC		CMTB	PartialGC	
KeyedDB 64	72 ± 2%	8.3 ± 5%	8.7x	290 ± 2%	26 ± 2%	11x
KeyedDB 128	140 ± 2%	9.5 ± 4%	15x	580 ± 2%	31 ± 3%	19x
KeyedDB 256	270 ± 1%	12 ± 6%	23x	1200 ± 3%	38 ± 5%	32x
MatrixMult8x8	110 ± 8%	100 ± 7%	1.1x	400 ± 10%	370 ± 5%	1.1x
Edit Distance 128	47 ± 7%	50 ± 9%	0.94x	120 ± 9%	180 ± 6%	0.67x
Millionaires 8192	140 ± 2%	20 ± 2%	7.0x	580 ± 1%	70 ± 2%	8.3x

In seconds

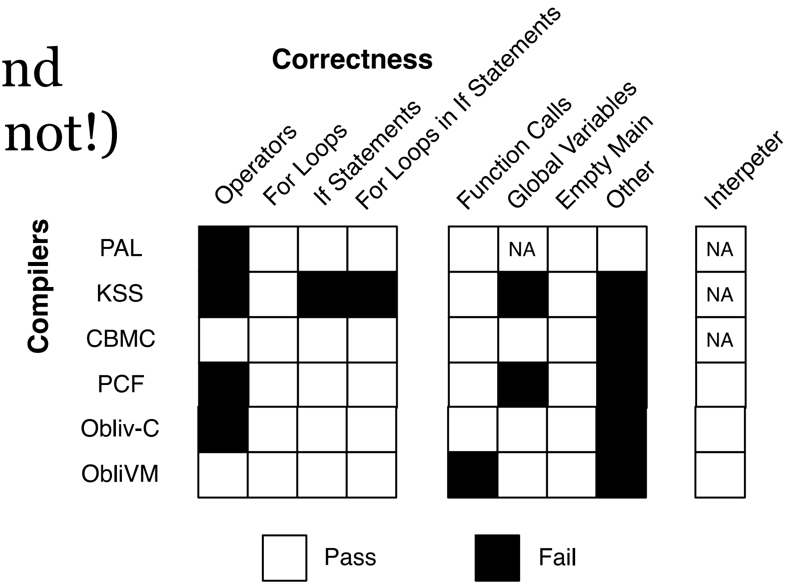
* both systems evaluated on same hardware, security parameters, and setup

Circuit Compilation



```
int add()
{
      a = input1 + input2;
      return a;
}
```

Goal: circuits need to be small and importantly, correct (many are not!)



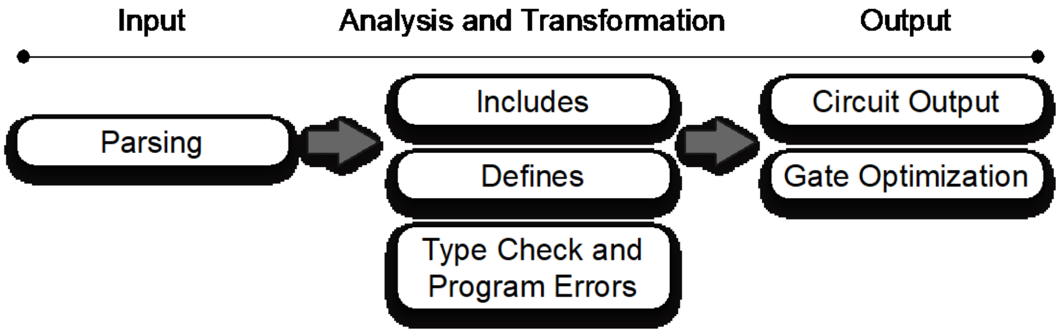
Correctness

Compilers	Operators	For Loops	If Statements	For Loops in If Statements	Function Calls	Global Variables	Empty Main	Other	Interpeter
PAL	■	☐	☐	☐	☐	NA	☐	☐	NA
KSS	■	☐	■	■	☐	■	☐	■	NA
CBMC	☐	☐	☐	☐	☐	☐	☐	■	NA
PCF	■	☐	☐	☐	☐	■	☐	■	☐
Obliv-C	■	☐	☐	☐	☐	☐	☐	■	☐
ObliVM	☐	☐	☐	☐	■	☐	☐	■	☐

☐ Pass ■ Fail

Frigate Compiler

- In response, we created Frigate
- Used standard compiler practices
  - Validation testing
  - Proper data structures, e.g., AST
  - Created a formal description of how operations should function



Add

$$\textsc{unc-Call}\quad \frac{\Gamma \vdash t_i : T_i \quad f : F}{\Gamma \vdash f(t_0 \ldots t_{n-1}) : R}$$

Performance

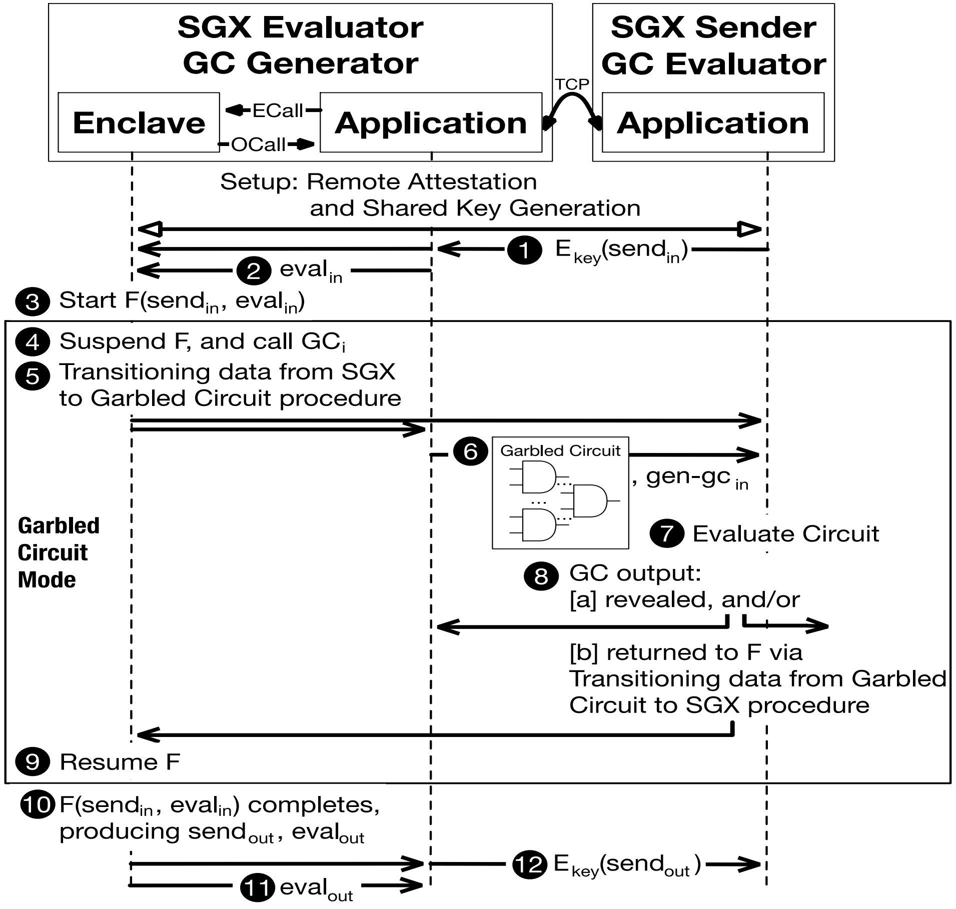
- Compared gate-counts with TinyGarble (writing circuits directly in Verilog and C)

	Frigate	TinyGarble	
ProgramName		C	Verlog
Hamming-160	719	1,264	158
Sum-1024	1,025	3,067	1,023
Compare-16384	16,386	52,224	16,384
X-to-X-bit Mult-64	4,035	-	3,925
MatrixMult5x5	128,252	-	120,125
AES	10,383	-	5,760

- Future directions: formal validation (e.g., translation validation) – full certification (e.g., CompCert) probably a ways out

Hardware-Assisted SMC with SGX

Similar guarantees to strictly garbled-circuit evaluation but dramatically faster



SGX Evaluator GC Generator

Enclave ← ECall — **Application**

— OCall →

TCP

SGX Sender GC Evaluator

Application

Setup: Remote Attestation and Shared Key Generation

① $E_{key}(send_{in})$

② $eval_{in}$

③ Start F($send_{in}$, $eval_{in}$)

④ Suspend F, and call GC_i

⑤ Transitioning data from SGX to Garbled Circuit procedure

⑥ Garbled Circuit, gen-gc $_{in}$

⑦ Evaluate Circuit

⑧ GC output: [a] revealed, and/or

[b] returned to F via Transitioning data from Garbled Circuit to SGX procedure

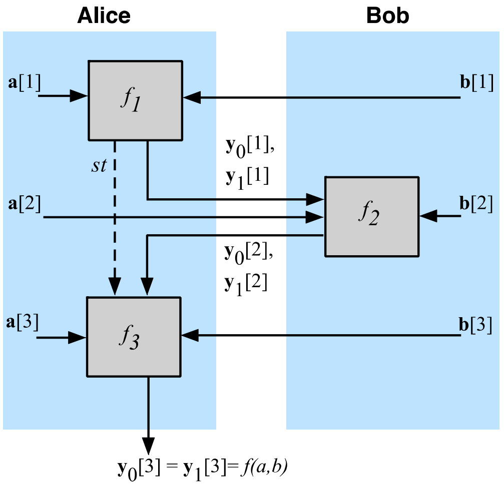
Garbled Circuit Mode

⑨ Resume F

⑩ F($send_{in}$, $eval_{in}$) completes, producing $send_{out}$, $eval_{out}$

⑪ $eval_{out}$

⑫ $E_{key}(send_{out})$



```
$\mathrm{Exec}_{\mathcal{P},k}(a, b)$
$\mathbf{a} \leftarrow_\$ \mathrm{SpA}(1^k, a)$; $\mathbf{b} \leftarrow_\$ \mathrm{SpB}(1^k, b)$; $st, \mathbf{y}_0[0], \mathbf{y}_1[0] \leftarrow \varepsilon$
for $j \leftarrow 1$ to $\ell$ do
    $u \leftarrow \mathbf{a}[j] \,\|\, \mathbf{y}_0[j-1]$; $v \leftarrow \mathbf{b}[j] \,\|\, \mathbf{y}_1[j-1]$
    if $j$ is odd then $(\mathbf{y}_0[j], \mathbf{y}_1[j], st) \leftarrow f_j(u, v, st)$
    else $(\mathbf{y}_0[j], \mathbf{y}_1[j]) \leftarrow f_j(u, v)$
return $(\mathbf{a}, \mathbf{b}, \mathbf{y}_0, \mathbf{y}_1)$
```

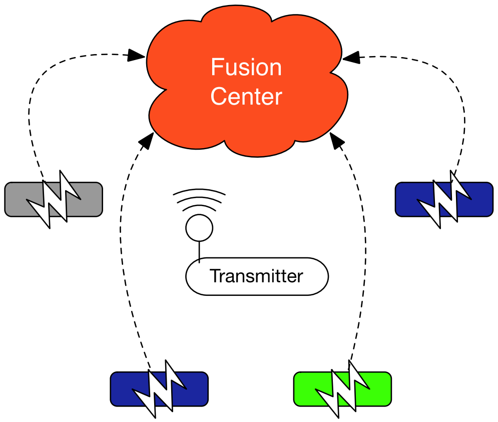
3-way even-odd partitioning of function f; f1, f3 computed in SGX enclave, f2 computed via garbling schemes and OT

Privacy-Preserving Localization

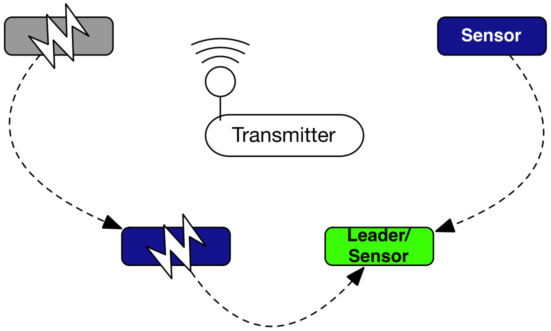
Algorithm 1: Particle Filter-based Localization: Main function

```
Function main(treeDepth, sensors, latPrecis, longPrecis,
  leafParticleCount, pleRange):
    for treeLevel = treeDepth; treeLevel > 0; treeLevel− do
        for sensor ∈ sensors[treeLevel] do
            if treeLevel == treeDepth then
                sensor.particles = leafInit(leafParticleCount,
                  pleRange)
            end
            else
                sensor.particles =
                  receiveParticles(sensor.child1, sensor.child2)
            end
            sensor.particles =
              updateParticles(sensor.particles, sensor.RSS)
        end
    end
    grid = ⟨maxLat, minLat, maxLong, minLong⟩ from
      root.particles
    while (maxLat − minLat > latPrecis) and
      (maxLong − minLong > longPrecis) do
        rootParticles = partitionParticles(rootParticles, grid)
        recalculate grid from root.particles
    end
    estimate = ⟨latitude, longitude⟩ at center of rootParticles
    return estimate
```

Centralized



Decentralized



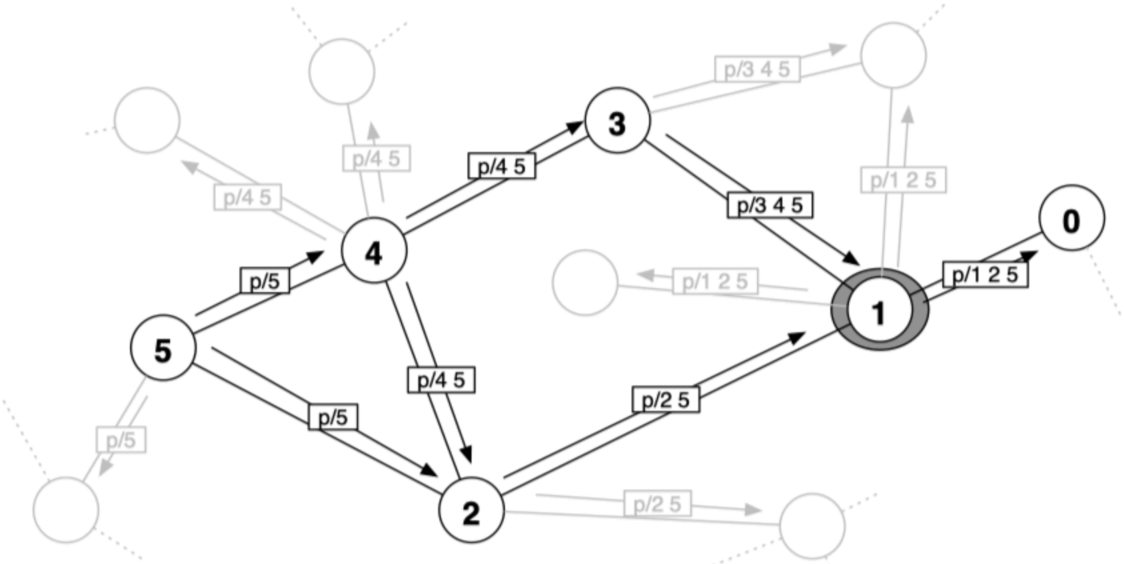
Compute particle measurement in a secure enclave – allows for remote attestation of the enclave and assurance of integrity

To-Do: Incorporate for low-power environments, incorporate hybrid circuit protocols

Open Challenges

- More efficient and optimized algorithms for SMC particularly in the malicious model
  - And for resource-constrained devices
- Algorithms for adapting hardware-assisted SMC to resource-constrained platforms
  - Formal assurances from TEEs and other enclave-based mechanisms on low-powered devices
- Move beyond Yao to other 2PC models (e.g. GMW)
- Assuring secure communication through highly-efficient cryptography
- Consider trustworthy platforms, e.g., running secure communication on seL4 kernels

Path Authentication in Routing Networks

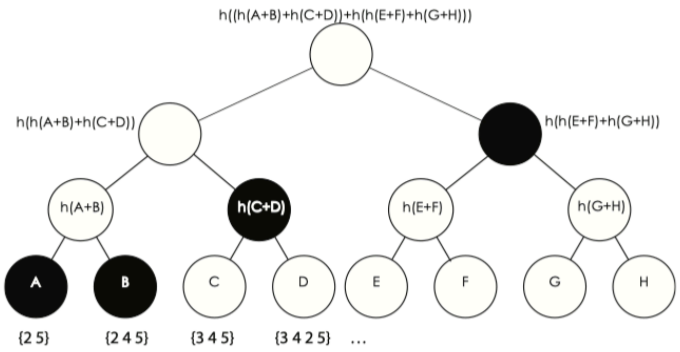


$$[[[[[P,\{5\},t_{n-2}]_{S_5}]P,\{2\ 5\},t_{n-1}]_{S_2}]P,\{1\ 2\ 5\},t_n]_{S_1}.$$

Example: path announcement for Internet routes using BGP

$$\left[\begin{array}{l} P,\{2\ 5\},h^{365}(x_1) \\ P,\{2\ 4\ 5\},h^{365}(x_2) \\ P,\{3\ 4\ 5\},h^{365}(x_3) \\ P,\{3\ 4\ 2\ 5\},h^{365}(x_4) \end{array}\right]_{S_1}$$

Signing each hop of the announcement (expensive signatures)



Hash chains/Merkle hash trees can vastly reduce computation overhead (set membership proofs)

UF UNIVERSITY of FLORIDA

Duke UNIVERSITY

TEXAS The University of Texas at Austin

UC SANTA CRUZ