Hashing Garbled Circuits for Free

Xiong Fan, **Chaya Ganesh** and Vladimir Kolesnikov

Motivation

- Garbled circuits (GC) – main technique for secure computation

Motivation

- Garbled circuits (GC) – main technique for secure computation
- Primitive in its own right

Motivation

- Garbled circuits (GC) – main technique for secure computation
- Primitive in its own right
- Hashing Garbled circuits

Motivation

- Garbled circuits (GC) – main technique for secure computation
- Primitive in its own right
- Hashing Garbled circuits
  - Cut-and-choose for GC-based 2PC

Motivation

- Garbled circuits (GC) – main technique for secure computation
- Primitive in its own right
- Hashing Garbled circuits
  - Cut-and-choose for GC-based 2PC
  - Private certified functions

Motivation

- Garbled circuits (GC) – main technique for secure computation
- Primitive in its own right
- Hashing Garbled circuits
  - Cut-and-choose for GC-based 2PC
  - Private certified functions
  - Encrypted database – Blind seer

Motivation

- Natural way – Generate GC, then hash

Motivation

- Natural way – Generate GC, then hash
- $GC = Garble(C), h = SHA(GC)$

Motivation

- Natural way – Generate GC, then hash
- $GC = Garble(C), h = SHA(GC)$
- Relative cost of fixed-key cipher garbling and hashing

Motivation

- Natural way – Generate GC, then hash
- $GC = Garble(C), h = SHA(GC)$
- Relative cost of fixed-key cipher garbling and hashing
- Fast hardware AES implementations, fast garbling, SHA bottleneck

Motivation

- Hashing GC costs up to 6$\times$ or more of GC generation

Motivation

- Hashing GC costs up to 6× or more of GC generation
- Free hash – hashing GC at no additional cost during GC generation

Motivation

- Hashing GC costs up to 6× or more of GC generation
- Free hash – hashing GC at no additional cost during GC generation
- Eliminating GC hashing cost significantly improves performance in GC applications

Private policy credentials

- Attribute-based credential

Private policy credentials

- Attribute-based credential
- Prover's input satisfies a certain policy

Private policy credentials

- Attribute-based credential
- Prover's input satisfies a certain policy
- Verifier's policy is private

Private policy credentials

- Attribute-based credential
- Prover's input satisfies a certain policy
- Verifier's policy is private
- Cut-and-choose approach reveals the policy function

Private policy credentials

- Attribute-based credential
- Prover's input satisfies a certain policy
- Verifier's policy is private
- Cut-and-choose approach reveals the policy function
- Certificate Authority (CA) setting – CA certifies the policy function

Certificate Authority (sk, vk)

Prover

Verifier

Certificate Authority (sk, vk)

$s_1, s_2, \ldots, s_n$

Prover

Verifier

Certificate Authority (sk, vk)

$s_1, s_2, \ldots, s_n$

$GC_i, d_i = \text{Garble}(f ; R_i)$

Prover

Verifier

Certificate Authority (sk, vk)

$s_1, s_2, \ldots, s_n$

Private policy function

$GC_i, d_i = \text{Garble}(f ; R_i)$

Prover

Verifier

Certificate Authority (sk, vk)

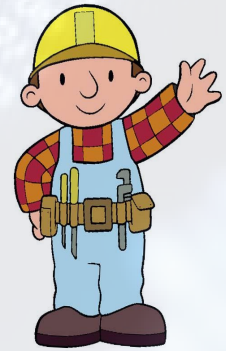
$s_1, s_2, \ldots, s_n$

Private policy function

$GC_i, d_i = \text{Garble}(f ; R_i)$

Prover

Randomness generated using s_i as seed



Verifier

Certificate Authority (sk, vk)

$s_1, s_2, \ldots, s_n$

$GC_i, d_i = \text{Garble}(f; R_i)$

$h_i = H(GC_i)$

Prover

Verifier

Certificate Authority (sk, vk)

$s_1, s_2, \ldots, s_n$

$GC_i, d_i = \text{Garble}(f ; R_i)$

$h_i = H(GC_i)$

$\sigma_i = \text{Sign}(h_i \| d_i, sk)$

Prover

Verifier

Certificate Authority (sk, vk)

$s_1, s_2, \ldots, s_n$

$GC_i, d_i = \text{Garble}(f; R_i)$

$h_i = H(GC_i)$

$\boldsymbol{\sigma}_i = \text{Sign}(h_i \,||\, d_i, sk)$

$(s_i, \boldsymbol{\sigma}_i)$

Verifier

Prover

Certificate Authority (sk, vk)

$s_1, s_2, \ldots, s_n$

$GC_i, d_i = \text{Garble}(f ; R_i)$

$h_i = H(GC_i)$

$\boldsymbol{\sigma}_i = \text{Sign}(h_i \,||\, d_i, sk)$

Prover

$(s_i, \boldsymbol{\sigma}_i)$

Verifier

$GC_i, d_i = \text{Garble}(f ; R_i)$

Certificate Authority (sk, vk)

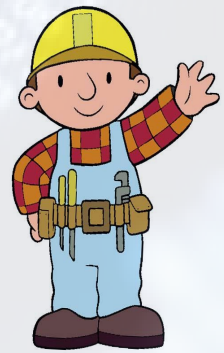
$s_1, s_2, \ldots, s_n$

$GC_i, d_i = \text{Garble}(f; R_i)$

$h_i = H(GC_i)$

$\boldsymbol{\sigma}_i = \text{Sign}(h_i \| d_i, sk)$

Prover



$(s_i, \boldsymbol{\sigma}_i)$

Verifier

$(GC_i, d_i, \boldsymbol{\sigma}_i)$

$GC_i, d_i = \text{Garble}(f; R_i)$

Certificate Authority (sk, vk)

$s_1, s_2, \ldots, s_n$

$GC_i, d_i = \text{Garble}(f; R_i)$

$h_i = H(GC_i)$

$\sigma_i = \text{Sign}(h_i \| d_i, sk)$

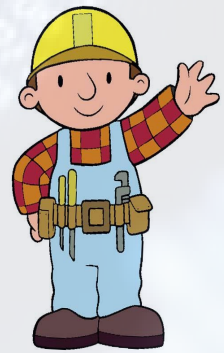
(s_i, σ_i)

Verifier

(GC_i, d_i, σ_i)

$GC_i, d_i = \text{Garble}(f; R_i)$

Prover



$h_i = H(GC_i)$

Certificate Authority (sk, vk)

$s_1, s_2, \ldots, s_n$

$GC_i, d_i = \text{Garble}(f; R_i)$

$h_i = H(GC_i)$

$\boldsymbol{\sigma}_i = \text{Sign}(h_i \| d_i, sk)$

$(s_i, \boldsymbol{\sigma}_i)$

Verifier

$(GC_i, d_i, \boldsymbol{\sigma}_i)$

$GC_i, d_i = \text{Garble}(f; R_i)$

Prover

$h_i = H(GC_i)$

If Verify $(h_i \| d_i, \boldsymbol{\sigma}_i, vk) \neq 1$, abort

Hashing in cut-and-choose

- Send hash of GCs in cut-and-choose protocols (GMS'08)

Hashing in cut-and-choose

- Send hash of GCs in cut-and-choose protocols (GMS'08)
- P_1 uses a seed s_i to construct GC_i

Hashing in cut-and-choose

- Send hash of GCs in cut-and-choose protocols (GMS'08)
- P_1 uses a seed s_i to construct GC_i
- Sends $h_1, \cdots h_n$, $h_i = H(GC_i)$

Hashing in cut-and-choose

- Send hash of GCs in cut-and-choose protocols (GMS'08)
- P_1 uses a seed s_i to construct GC_i
- Sends $h_1, \cdots h_n$, $h_i = H(GC_i)$
- If GC_i is a check circuit, reveal s_i

Hashing in cut-and-choose

- Send hash of GCs in cut-and-choose protocols (GMS'08)
- P_1 uses a seed s_i to construct GC_i
- Sends $h_1, \cdots h_n$, $h_i = H(GC_i)$
- If GC_i is a check circuit, reveal s_i
- P_2 reconstructs GC_i from s_i and verifies h_i for check circuit

Hashing in cut-and-choose

- Send hash of GCs in cut-and-choose protocols (GMS'08)
- P_1 uses a seed s_i to construct GC_i
- Sends $h_1, \cdots h_n$, $h_i = H(GC_i)$
- If GC_i is a check circuit, reveal s_i
- P_2 reconstructs GC_i from s_i and verifies h_i for check circuit
- Using a CR hash trades off computation for communication

Hashing in cut-and-choose

- Send hash of GCs in cut-and-choose protocols (GMS'08)
- P_1 uses a seed s_i to construct GC_i
- Sends $h_1, \cdots h_n$, $h_i = H(GC_i)$
- If GC_i is a check circuit, reveal s_i
- P_2 reconstructs GC_i from s_i and verifies h_i for check circuit
- Using a CR hash trades off computation for communication
- Can free hash be used instead?

Summary of results

- Definition of GC hash security

Summary of results

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]

Summary of results

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation

Summary of results

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation
- Impact – Applications of free hash

Garbling scheme

Tuple of algorithms (Garble, Encode, Eval, Decode)

- $\text{Garble}(C) = (\hat{C}, e, d)$
- $\text{Encode}(x, e) = \hat{x}$
- $\text{Eval}(\hat{C}, \hat{x}) = \hat{z}$
- $\text{Decode}(\hat{z}, d) = z$
- Security properties:
  - Correctness: $z = C(x)$

Garbling scheme

Tuple of algorithms (Garble, Encode, Eval, Decode)

- $\text{Garble}(C) = (\hat{C}, e, d)$
- $\text{Encode}(x, e) = \hat{x}$
- $\text{Eval}(\hat{C}, \hat{x}) = \hat{z}$
- $\text{Decode}(\hat{z}, d) = z$
- Security properties:
  - Correctness: $z = C(x)$
  - Privacy: $(\hat{C}, \hat{x}, d)$ reveals nothing beyond $C(x)$

Garbling scheme

Tuple of algorithms (Garble, Encode, Eval, Decode)

- $\text{Garble}(C) = (\hat{C}, e, d)$
- $\text{Encode}(x, e) = \hat{x}$
- $\text{Eval}(\hat{C}, \hat{x}) = \hat{z}$
- $\text{Decode}(\hat{z}, d) = z$
- Security properties:
  - Correctness: $z = C(x)$
  - Privacy: $(\hat{C}, \hat{x}, d)$ reveals nothing beyond $C(x)$
  - Authenticity: given $(\hat{C}, \hat{x})$, hard to find z' such that $\mathit{decode}(z', d) \notin \{C(x), \perp\}$

Garbling scheme

Tuple of algorithms (Garble, Encode, Eval, Decode)

- $\text{Garble}(C) = (\hat{C}, e, d)$
- $\text{Encode}(x, e) = \hat{x}$
- $\text{Eval}(\hat{C}, \hat{x}) = \hat{z}$
- $\text{Decode}(\hat{z}, d) = z$
- Security properties:
  - Correctness: $z = C(x)$
  - Privacy: $(\hat{C}, \hat{x}, d)$ reveals nothing beyond $C(x)$
  - Authenticity: given $(\hat{C}, \hat{x})$, hard to find z' such that $\mathit{decode}(z', d) \notin \{C(x), \perp\}$
  - Verifiability: Additional algorithm Ve, $\text{Ve}(C, \hat{C}, e, d) \in \{0, 1\}$

Overview

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation
- Impact – Applications of free hash

Overview

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation
- Impact – Applications of free hash

GC hash definition

- Take advantage of the input to hash being a Garbled Circuit

GC hash definition

- Take advantage of the input to hash being a Garbled Circuit
- GC hash definition weaker than standard collision resistance

GC hash definition

- Take advantage of the input to hash being a Garbled Circuit
- GC hash definition weaker than standard collision resistance
- Given a correctly generated garbled circuit and hash (GC, h)

GC hash definition

- Take advantage of the input to hash being a Garbled Circuit
- GC hash definition weaker than standard collision resistance
- Given a correctly generated garbled circuit and hash (GC, h)
  - If $\mathcal{A}$ finds $\widehat{GC}$ such that $H(\widehat{GC}) = H(GC)$

GC hash definition

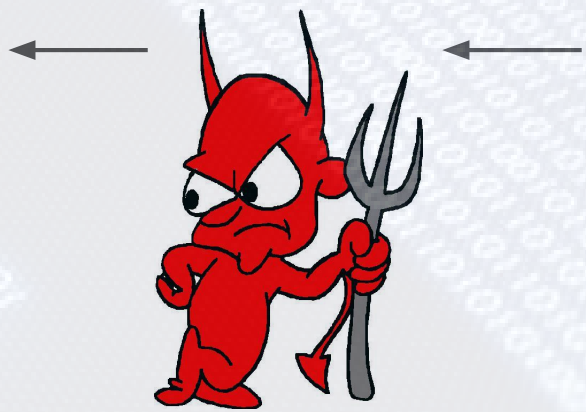
- Take advantage of the input to hash being a Garbled Circuit
- GC hash definition weaker than standard collision resistance
- Given a correctly generated garbled circuit and hash (GC, h)
  - If $\mathcal{A}$ finds $\widehat{GC}$ such that $H(\widehat{GC}) = H(GC)$
  - Then, w.h.p, the garbled circuit property of $\widehat{GC}$ is broken

GC hash definition

- Take advantage of the input to hash being a Garbled Circuit
- GC hash definition weaker than standard collision resistance
- Given a correctly generated garbled circuit and hash (GC, h)
  - If $\mathcal{A}$ finds $\widehat{GC}$ such that $H(\widehat{GC}) = H(GC)$
  - Then, w.h.p, the garbled circuit property of $\widehat{GC}$ is broken
  - $\widehat{GC}$ will fail to evaluate

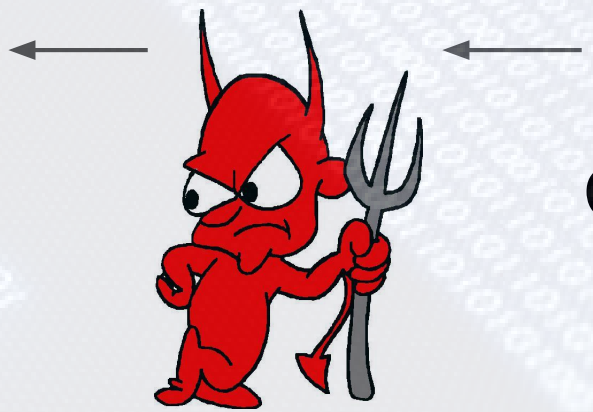
C

GC, $\mathcal{GC}$, e, e, d, h



C

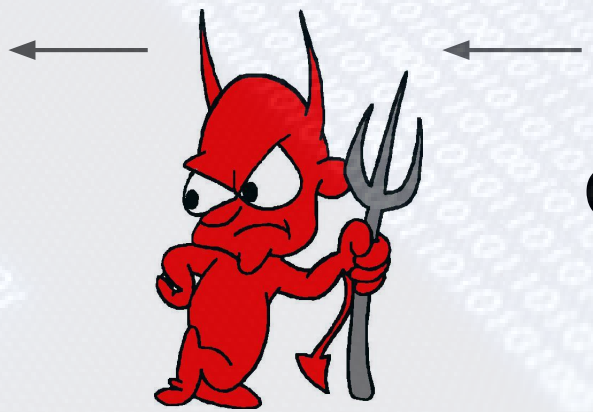
GC, $\mathcal{GC}$, e, e, d, h



C

H(GC) = H($\mathcal{GC}$) = h

GC, $\mathcal{GC}$, e, e, d, h

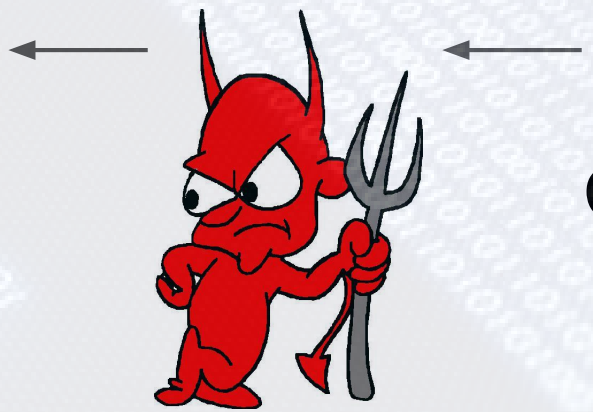


C

H(GC) = H($\mathcal{GC}$) = h

Ve(C, GC, d, e) = accept

GC, $\mathcal{GC}$, e, e, d, h



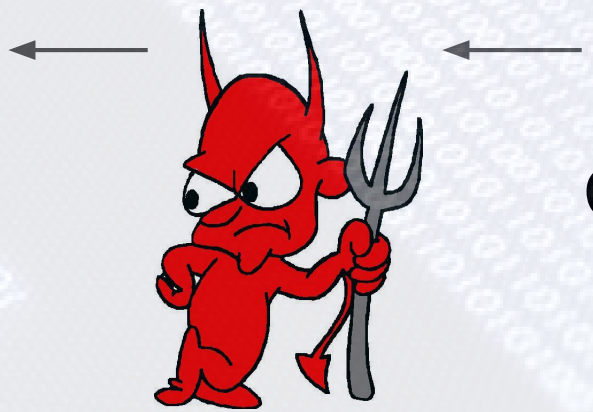
C

H(GC) = H($\mathcal{GC}$) = h

Ve(C, GC, d, e) = accept

De(Eval($\mathcal{GC}$, En(e, x), d)) = $\perp$ for all x , w.h.p

GC, $\mathcal{GC}$, e, e, d, h



C

H(GC) = H($\mathcal{GC}$) = h

Ve(C, GC, d, e) = accept

De(Eval($\mathcal{GC}$, En(e, x), d)) = $\perp$ for all x , w.h.p

GC, $\mathcal{GC}$, e, e, d, h

C

H(GC) = H($\mathcal{GC}$) = h

Same decoding information d

Ve(C, GC, d, e) = accept

De(Eval($\mathcal{GC}$, En(e, x), d)) = $\perp$ for all x , w.h.p

Overview

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation
- Impact – Applications of free hash

Overview

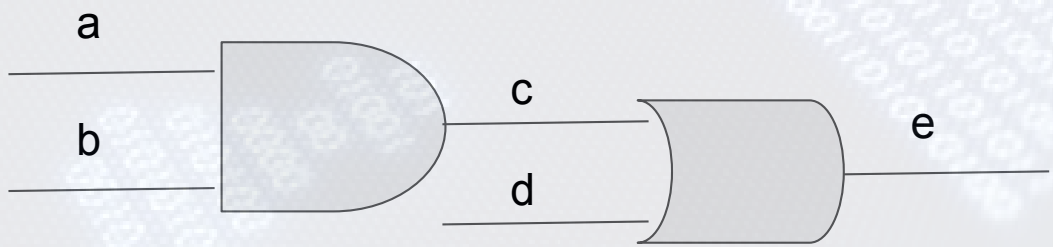
- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation
- Impact – Applications of free hash

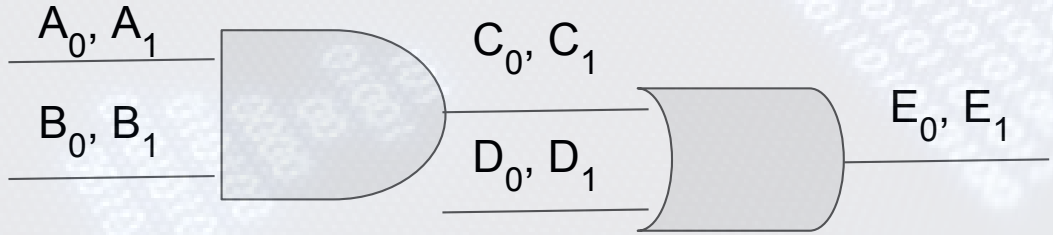
GC hash construction

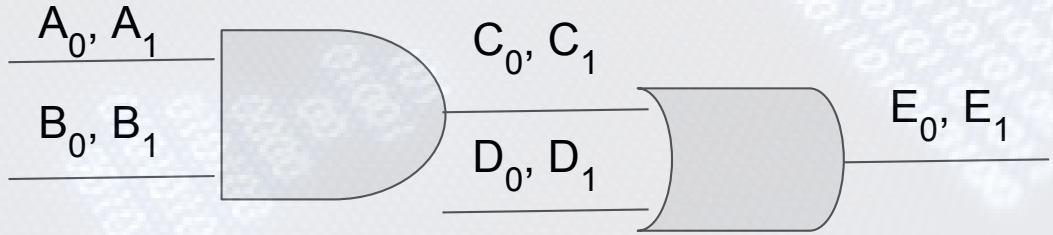
- Intertwine hash generation and verification with GC generation and evaluation

GC hash construction

- Intertwine hash generation and verification with GC generation and evaluation
- Attempt 1: $H(GC) = \oplus_i GR_i$

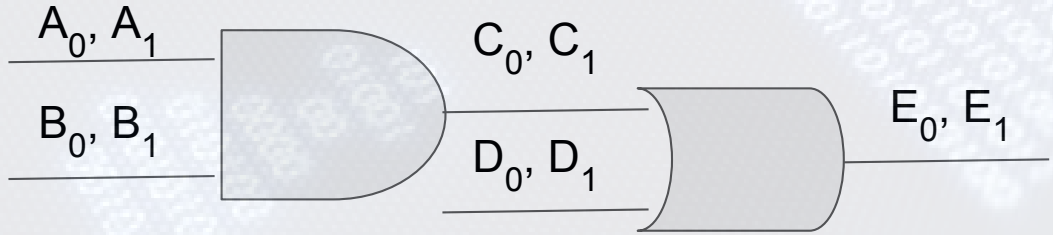

a
b
c
d
e


A_0, A_1
B_0, B_1
C_0, C_1
D_0, D_1
E_0, E_1



GT1

$E_{A0, B0}\ (C_0)$
$E_{A0, B1}\ (C_0)$
$E_{A1, B0}\ (C_0)$
$E_{A1, B1}\ (C_1)$

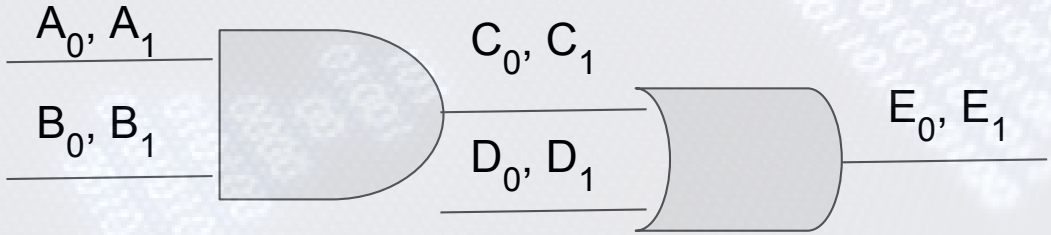


GT1

$E_{A0, B0}\ (C_0)$
$E_{A0, B1}\ (C_0)$
$E_{A1, B0}\ (C_0)$
$E_{A1, B1}\ (C_1)$

GT2

$E_{C0, D0}\ (E_0)$
$E_{C0, D1}\ (E_1)$
$E_{C1, D0}\ (E_1)$
$E_{C1, D1}\ (E_1)$



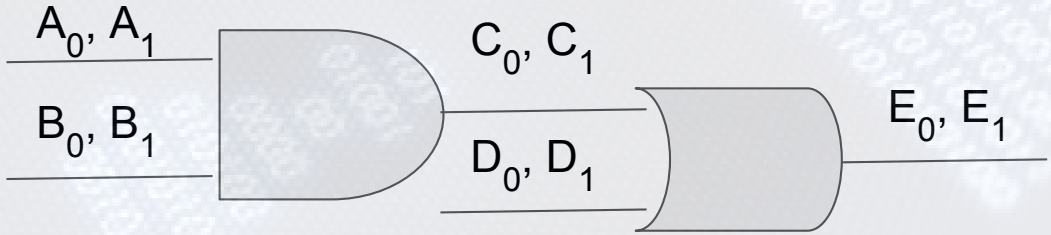
GT1

$E_{A0, B0}\ (C_0)$
$E_{A0, B1}\ (C_0)$
$E_{A1, B0}\ (C_0)$
$E_{A1, B1}\ (C_1)$

GT2

$E_{C0, D0}\ (E_0)$
$E_{C0, D1}\ (E_1)$
$E_{C1, D0}\ (E_1)$
$E_{C1, D1}\ (E_1)$

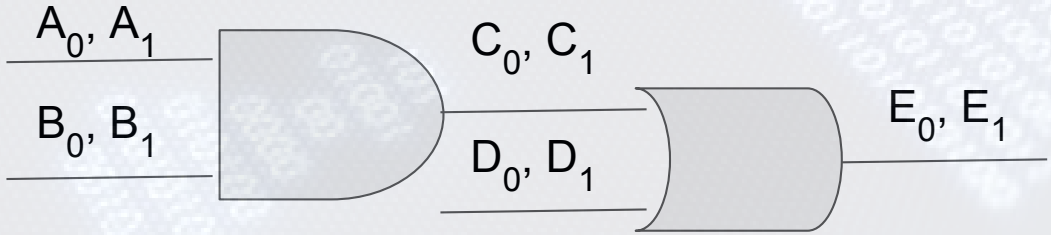
GC = (GT1, GT2)



GT1 GT2

$$h = \begin{array}{|c|}\hline E_{A0,B0}(C_0) \\ \hline E_{A0,B1}(C_0) \\ \hline E_{A1,B0}(C_0) \\ \hline E_{A1,B1}(C_1) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,D0}(E_0) \\ \hline E_{C0,D1}(E_1) \\ \hline E_{C1,D0}(E_1) \\ \hline E_{C1,D1}(E_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

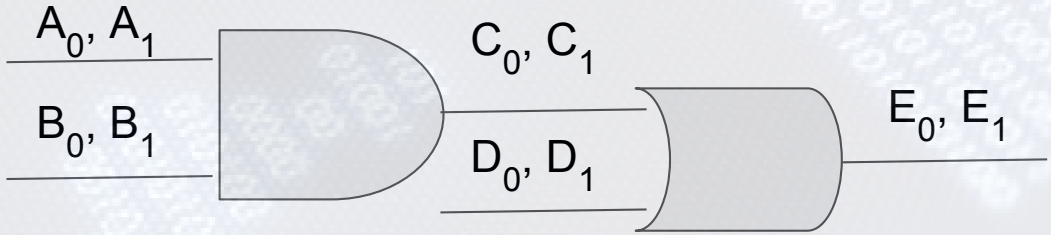


GT1 GT2

$$h = \begin{array}{|c|}\hline E_{A0,B0}(C_0) \\ \hline E_{A0,B1}(C_0) \\ \hline E_{A1,B0}(C_0) \\ \hline E_{A1,B1}(C_1) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,D0}(E_0) \\ \hline E_{C0,D1}(E_1) \\ \hline E_{C1,D0}(E_1) \\ \hline E_{C1,D1}(E_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

H(GC) = h

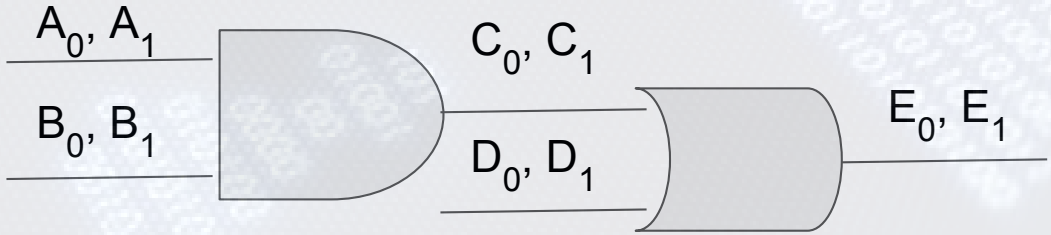


GT1 GT2

$$h = \begin{array}{|c|} \hline E_{A0,B0}(C_0) \\ \hline E_{A0,B1}(C_0) \\ \hline E_{A1,B0}(C_0) \\ \hline E_{A1,B1}(C_0) \\ \hline \end{array} \oplus \begin{array}{|c|} \hline E_{C0,D0}(E_0) \\ \hline E_{C0,D1}(E_1) \\ \hline E_{C1,D0}(E_1) \\ \hline E_{C1,D1}(E_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

H(GC) = h



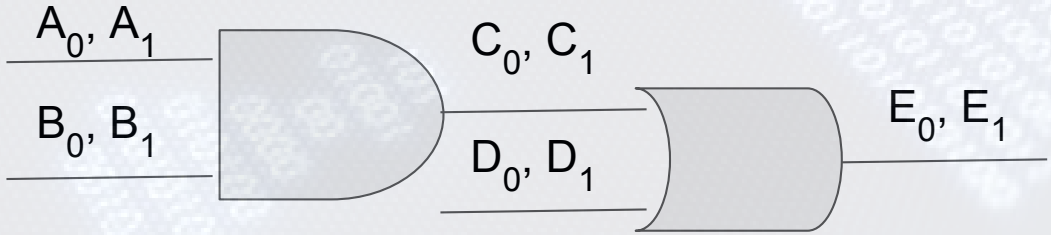
GT1 GT2

$$h = \begin{array}{|c|}\hline E_{A0,B0}(C_0) \\ \hline E_{A0,B1}(C_0) \\ \hline E_{A1,B0}(C_0) \\ \hline E_{A1,B1}(C_0) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,D0}(E_0) \\ \hline E_{C0,D1}(E_1) \\ \hline E_{C1,D0}(E_1) \\ \hline E_{C1,D1}(E_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

H(GC) = h

ĜC = (ĜT1, GT2)



GT1 GT2

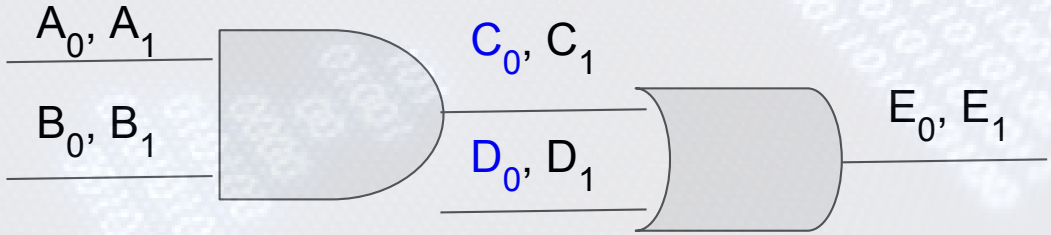
$$h = \begin{array}{|c|}\hline E_{A0,B0}(C_0) \\ \hline E_{A0,B1}(C_0) \\ \hline E_{A1,B0}(C_0) \\ \hline E_{A1,B1}(C_0) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,D0}(E_0) \\ \hline E_{C0,D1}(E_1) \\ \hline E_{C1,D0}(E_1) \\ \hline E_{C1,D1}(E_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

H(GC) = h

$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h \oplus \Delta$



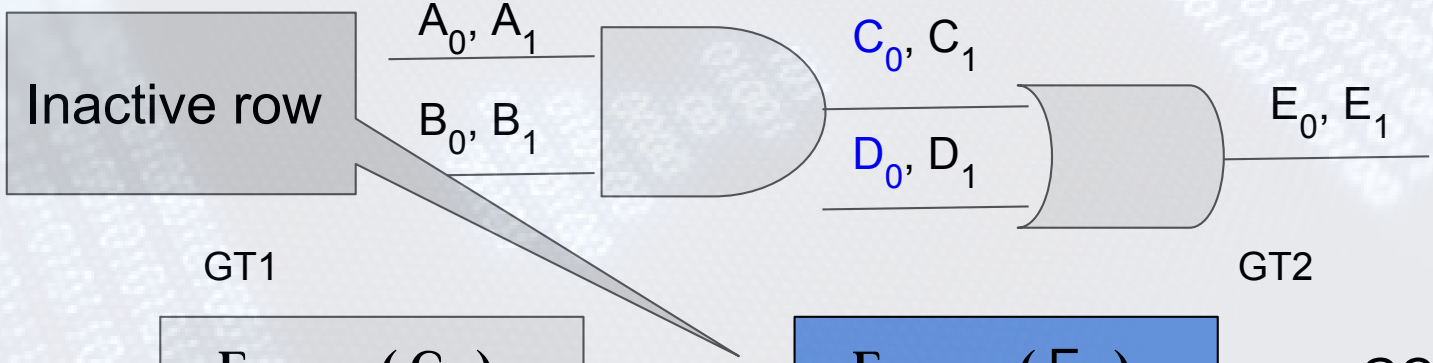
$$h = \begin{array}{|c|}\hline E_{A0,B0}(C_0) \\ \hline E_{A0,B1}(C_0) \\ \hline E_{A1,B0}(C_0) \\ \hline E_{A1,B1}(C_0) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,D0}(E_0) \\ \hline E_{C0,D1}(E_1) \\ \hline E_{C1,D0}(E_1) \\ \hline E_{C1,D1}(E_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

H(GC) = h

$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h \oplus \Delta$



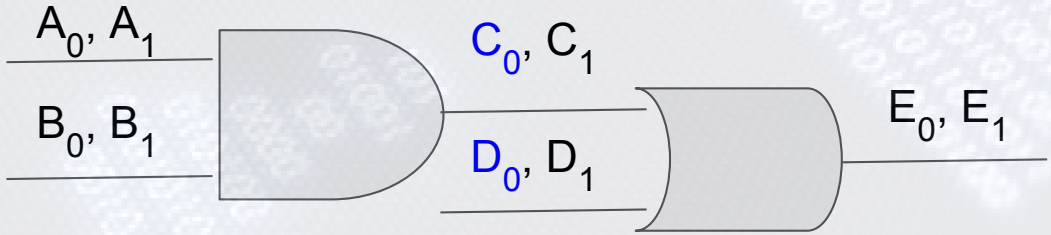
$$h = \begin{matrix} E_{A0,B0}(C_0) \\ E_{A0,B1}(C_0) \\ E_{A1,B0}(C_0) \\ E_{A1,B1}(C_0) \end{matrix} \oplus \begin{matrix} E_{C0,D0}(E_0) \\ E_{C0,D1}(E_1) \\ E_{C1,D0}(E_1) \\ E_{C1,D1}(E_1) \end{matrix}$$

GC = (GT1, GT2)

H(GC) = h

$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h \oplus \Delta$



GT1 GT2

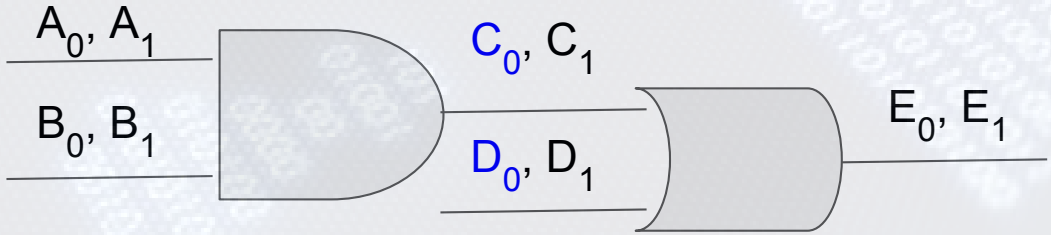
$$h = \begin{array}{|c|} \hline E_{A0,B0}(C_0) \\ \hline E_{A0,B1}(C_0) \\ \hline E_{A1,B0}(C_0) \\ \hline E_{A1,B1}(C_0) \\ \hline \end{array} \oplus \begin{array}{|c|} \hline E_{C0,D0}(E_0) \oplus \Delta \\ \hline E_{C0,D1}(E_1) \\ \hline E_{C1,D0}(E_1) \\ \hline E_{C1,D1}(E_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

H(GC) = h

$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h \oplus \Delta$



GT1 GT2

$h =$

$E_{A0, B0}(C_0)$	$E_{C0, D0}(E_0) \oplus \Delta$
$E_{A0, B1}(C_0)$	$E_{C0, D1}(E_1)$
$E_{A1, B0}(C_0)$	$E_{C1, D0}(E_1)$
$E_{A1, B1}(C_0)$	$E_{C1, D1}(E_1)$

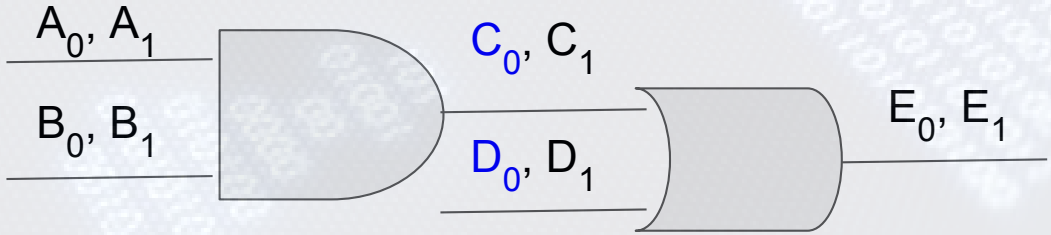
$\oplus$

GC = (GT1, GT2)

H(GC) = h

$\hat{GC}$ = ($\hat{GT1}$, GT2)

H($\hat{GC}$) = $h \oplus \Delta \oplus \Delta$



$$h = \begin{array}{|c|}\hline E_{A0,\,B0}\,(C_0) \\ \hline E_{A0,\,B1}\,(C_0) \\ \hline E_{A1,\,B0}\,(C_0) \\ \hline E_{A1,\,B1}\,(C_0) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,\,D0}(E_0) \oplus \Delta \\ \hline E_{C0,\,D1}\,(E_1) \\ \hline E_{C1,\,D0}\,(E_1) \\ \hline E_{C1,\,D1}\,(E_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

H(GC) = h

$\hat{GC}$ = ($\hat{GT1}$, GT2)
H($\hat{GC}$) = h ✔

GC hash construction

- Make each gate's output wire label depend on *all* entries of GT

GC hash construction

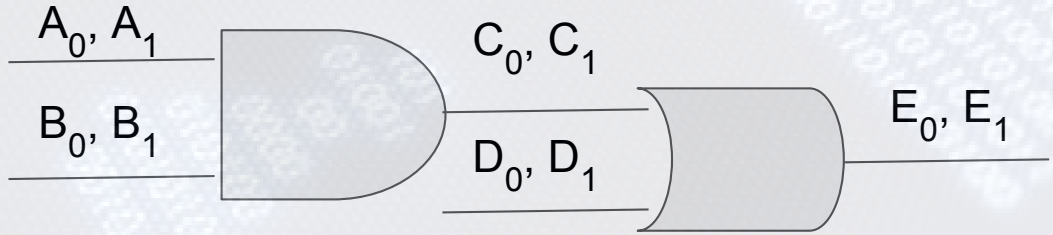
- Make each gate's output wire label depend on *all* entries of GT
- XOR hash correction involves modifying an active GT entry

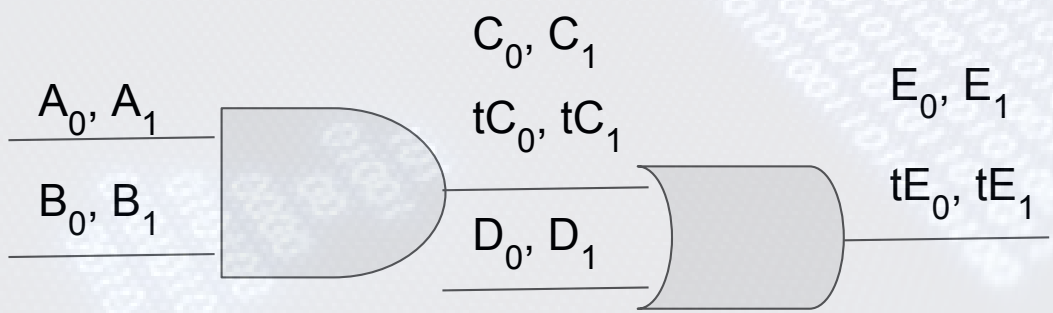
GC hash construction

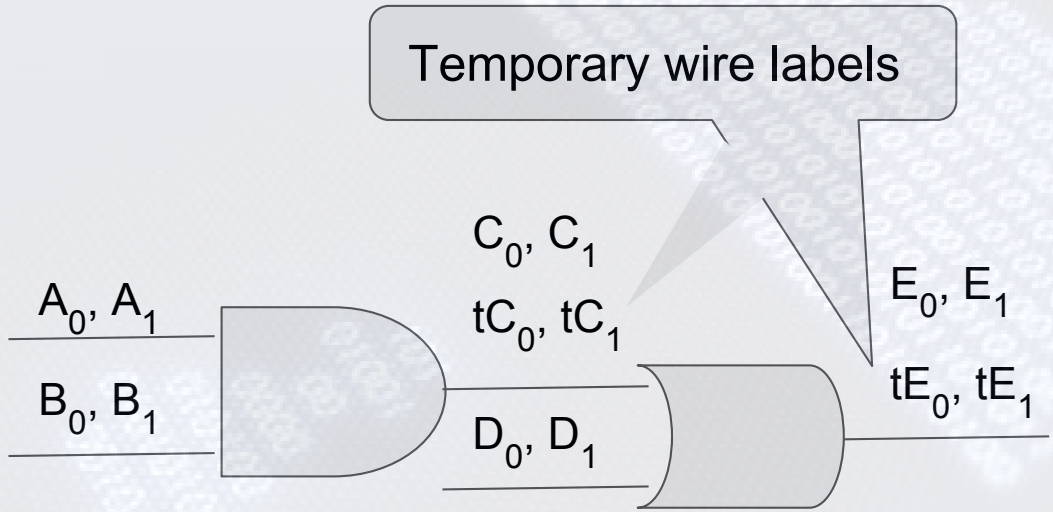
- Make each gate's output wire label depend on *all* entries of GT
- XOR hash correction involves modifying an active GT entry
- This affects the computed output wire label of the gate

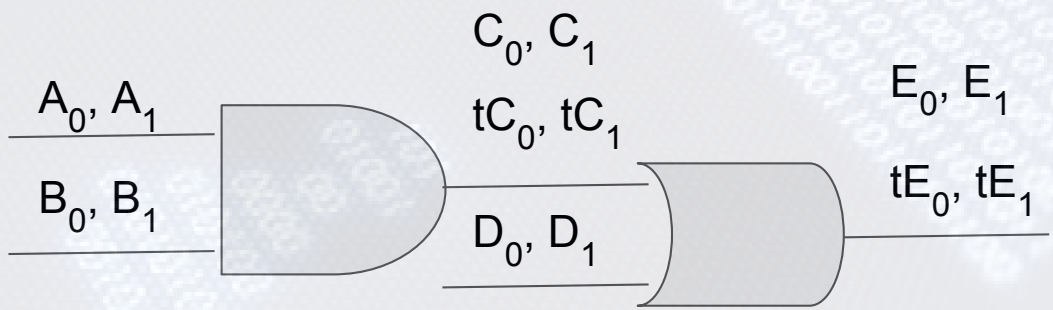
GC hash construction

- Make each gate's output wire label depend on *all* entries of GT
- XOR hash correction involves modifying an active GT entry
- This affects the computed output wire label of the gate
- Does this suffice?


A_0, A_1
B_0, B_1
C_0, C_1
D_0, D_1
E_0, E_1


A_0, A_1
B_0, B_1
C_0, C_1
tC_0, tC_1
D_0, D_1
E_0, E_1
tE_0, tE_1


Temporary wire labels
A_0, A_1
B_0, B_1
C_0, C_1
tC_0, tC_1
D_0, D_1
E_0, E_1
tE_0, tE_1



GT1

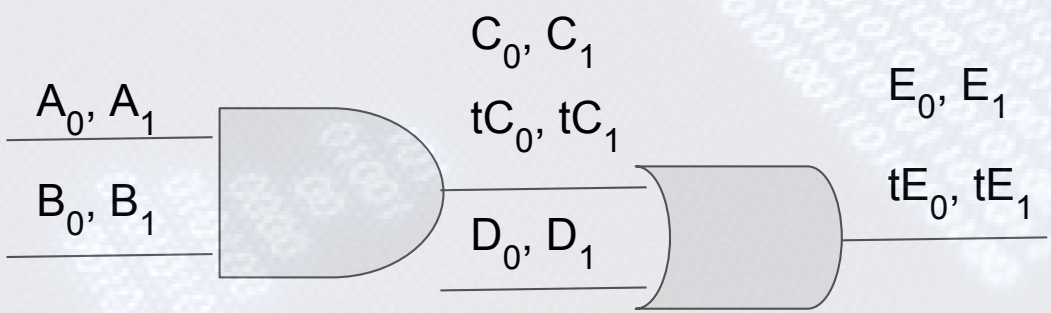
$E_{A0, B0}\ (tC_0)$
$E_{A0, B1}\ (tC_0)$
$E_{A1, B0}\ (tC_0)$
$E_{A1, B1}\ (tC_1)$

GT2

$E_{C0, D0}\ (tE_0)$
$E_{C0, D1}\ (tE_1)$
$E_{C1, D0}\ (tE_1)$
$E_{C1, D1}\ (tE_1)$

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



GT1

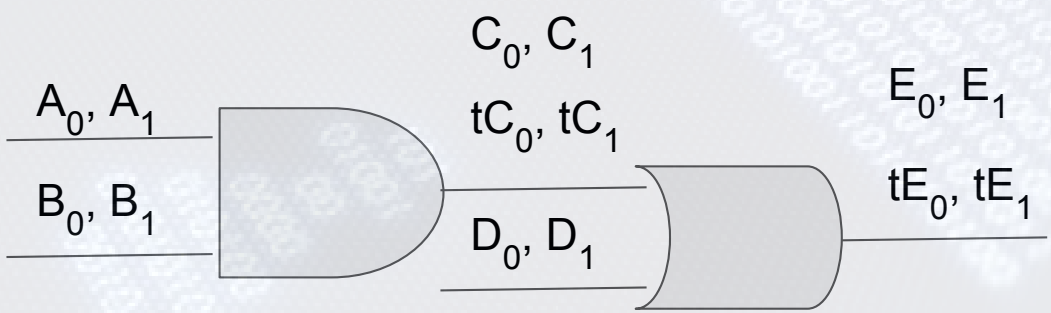
$E_{A0, B0}\ (tC_0)$
$E_{A0, B1}\ (tC_0)$
$E_{A1, B0}\ (tC_0)$
$E_{A1, B1}\ (tC_1)$

GT2

$E_{C0, D0}\ (tE_0)$
$E_{C0, D1}\ (tE_1)$
$E_{C1, D0}\ (tE_1)$
$E_{C1, D1}\ (tE_1)$

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



GT1 GT2

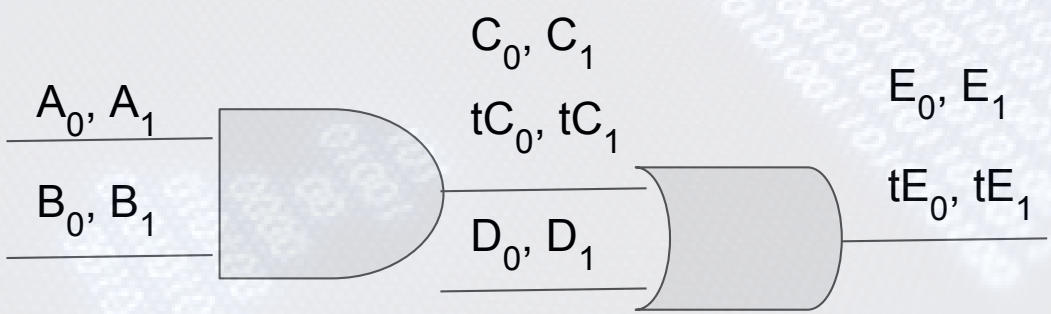
$$h = \begin{array}{|c|}\hline E_{A0,B0}(tC_0) \\ \hline E_{A0,B1}(tC_0) \\ \hline E_{A1,B0}(tC_0) \\ \hline E_{A1,B1}(tC_1) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,D0}(tE_0) \\ \hline E_{C0,D1}(tE_1) \\ \hline E_{C1,D0}(tE_1) \\ \hline E_{C1,D1}(tE_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

H(GC) = h

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



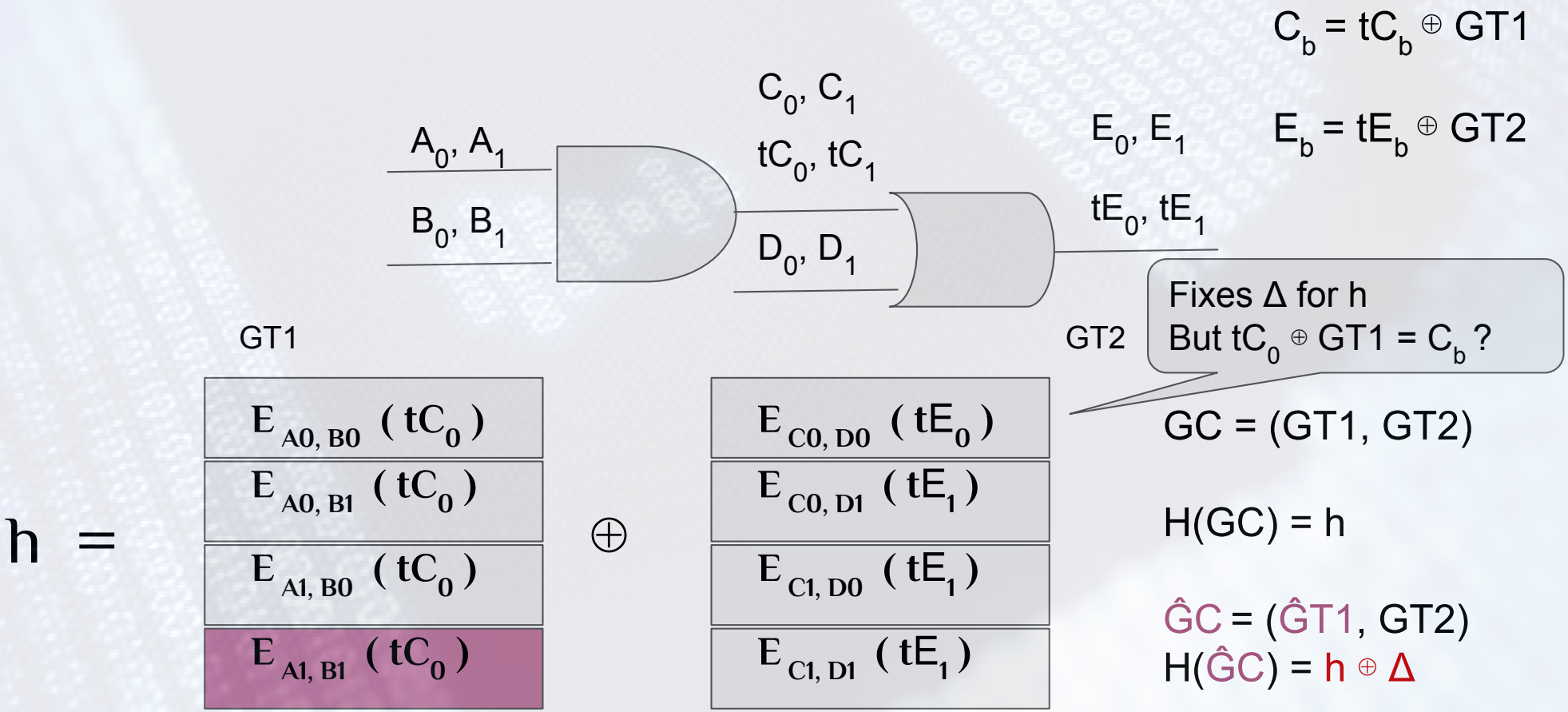
$$h = \begin{array}{|c|}\hline E_{A0,B0}\ (tC_0) \\ \hline E_{A0,B1}\ (tC_0) \\ \hline E_{A1,B0}\ (tC_0) \\ \hline E_{A1,B1}\ (tC_0) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,D0}\ (tE_0) \\ \hline E_{C0,D1}\ (tE_1) \\ \hline E_{C1,D0}\ (tE_1) \\ \hline E_{C1,D1}\ (tE_1) \\ \hline \end{array}$$

$GC = (GT1, GT2)$

$H(GC) = h$

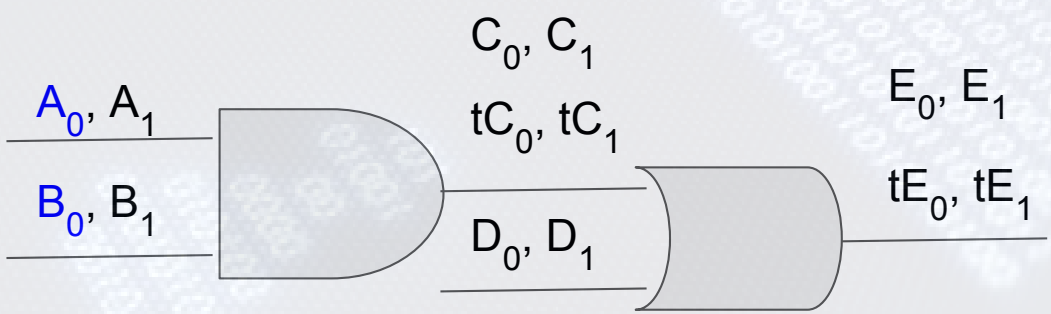
$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h \oplus \Delta$


$C_b = tC_b \oplus GT1$
$E_b = tE_b \oplus GT2$
A_0, A_1
B_0, B_1
C_0, C_1
tC_0, tC_1
D_0, D_1
E_0, E_1
tE_0, tE_1
Fixes Δ for h
But $tC_0 \oplus GT1 = C_b$?
GT1
GT2
$h =$
$E_{A0, B0}\ (tC_0)$
$E_{A0, B1}\ (tC_0)$
$E_{A1, B0}\ (tC_0)$
$E_{A1, B1}\ (tC_0)$
$\oplus$
$E_{C0, D0}\ (tE_0)$
$E_{C0, D1}\ (tE_1)$
$E_{C1, D0}\ (tE_1)$
$E_{C1, D1}\ (tE_1)$
GC = (GT1, GT2)
H(GC) = h
$\hat{GC} = (\hat{GT1}, GT2)$
$H(\hat{GC}) = h \oplus \Delta$

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



GT1 GT2

$$h = \begin{bmatrix} E_{A0,B0}(tC_0) \\ E_{A0,B1}(tC_0) \\ E_{A1,B0}(tC_0) \\ E_{A1,B1}(tC_0) \end{bmatrix} \oplus \begin{bmatrix} E_{C0,D0}(tE_0) \\ E_{C0,D1}(tE_1) \\ E_{C1,D0}(tE_1) \\ E_{C1,D1}(tE_1) \end{bmatrix}$$

$GC = (GT1, GT2)$

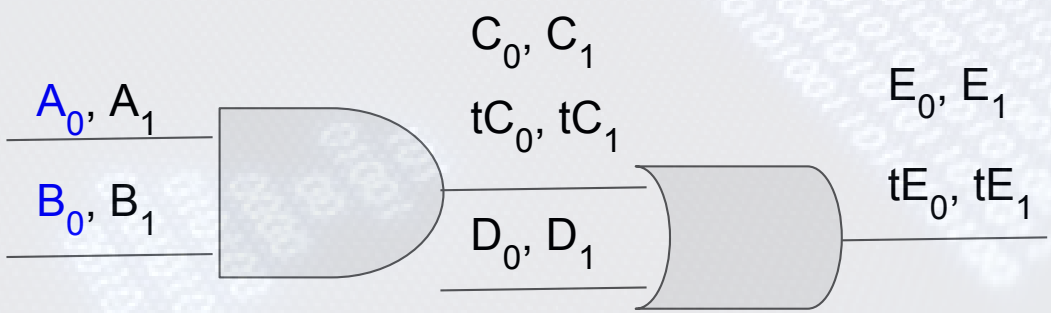
$H(GC) = h$

$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h \oplus \Delta$

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



GT1 GT2

$$h = \begin{array}{|c|}\hline E_{A0,B0}(tC_0) \\ \hline E_{A0,B1}(tC_0) \\ \hline E_{A1,B0}(tC_0) \\ \hline E_{A1,B1}(tC_0) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,D0}(tE_0) \\ \hline E_{C0,D1}(tE_1) \\ \hline E_{C1,D0}(tE_1) \\ \hline E_{C1,D1}(tE_1) \\ \hline \end{array}$$

$GC = (GT1, GT2)$

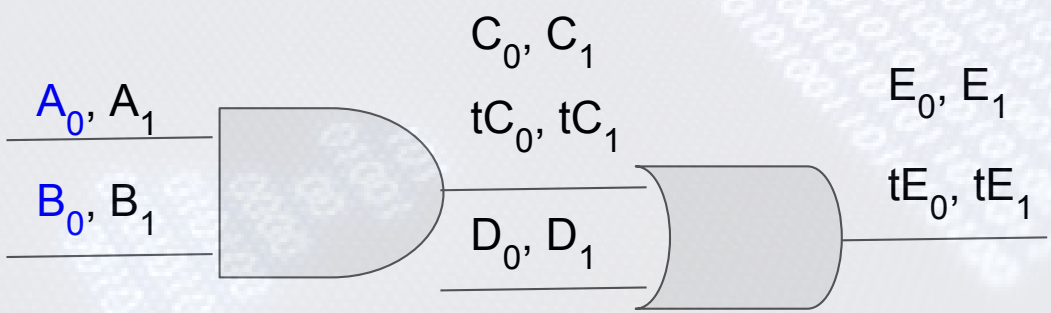
$H(GC) = h$

$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h \oplus \Delta$

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



GT1 GT2

$$h = \begin{array}{|l|}\hline E_{A0,B0}(tC_0) \oplus \Delta \\ \hline E_{A0,B1}(tC_0) \\ \hline E_{A1,B0}(tC_0) \\ \hline E_{A1,B1}(tC_0) \\ \hline \end{array} \oplus \begin{array}{|l|}\hline E_{C0,D0}(tE_0) \\ \hline E_{C0,D1}(tE_1) \\ \hline E_{C1,D0}(tE_1) \\ \hline E_{C1,D1}(tE_1) \\ \hline \end{array}$$

$GC = (GT1, GT2)$

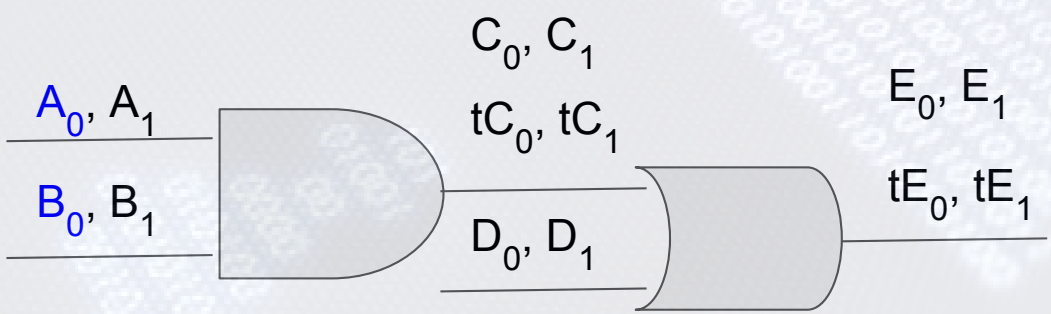
$H(GC) = h$

$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h \oplus \Delta$

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



GT1 GT2

$$h = \begin{array}{|l|}\hline E_{A0,B0}(tC_0) \oplus \Delta \\ \hline E_{A0,B1}(tC_0) \\ \hline E_{A1,B0}(tC_0) \\ \hline E_{A1,B1}(tC_0) \\ \hline \end{array} \oplus \begin{array}{|l|}\hline E_{C0,D0}(tE_0) \\ \hline E_{C0,D1}(tE_1) \\ \hline E_{C1,D0}(tE_1) \\ \hline E_{C1,D1}(tE_1) \\ \hline \end{array}$$

$GC = (GT1, GT2)$

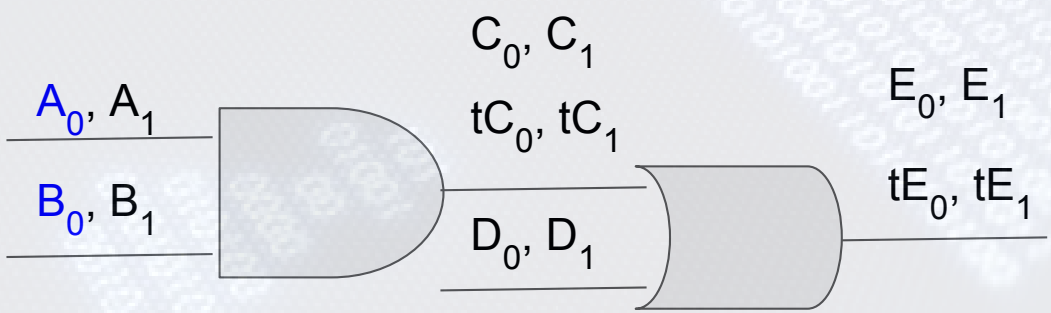
$H(GC) = h$

$\hat{G}C = (\hat{G}T1, GT2)$

$H(\hat{G}C) = h \oplus \Delta \oplus \Delta$

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



$$h = \begin{array}{|l|}\hline E_{A0,B0}(tC_0) \oplus \Delta \\ \hline E_{A0,B1}(tC_0) \\ \hline E_{A1,B0}(tC_0) \\ \hline E_{A1,B1}(tC_0) \\ \hline \end{array} \oplus \begin{array}{|l|}\hline E_{C0,D0}(tE_0) \\ \hline E_{C0,D1}(tE_1) \\ \hline E_{C1,D0}(tE_1) \\ \hline E_{C1,D1}(tE_1) \\ \hline \end{array}$$

$GC = (GT1, GT2)$

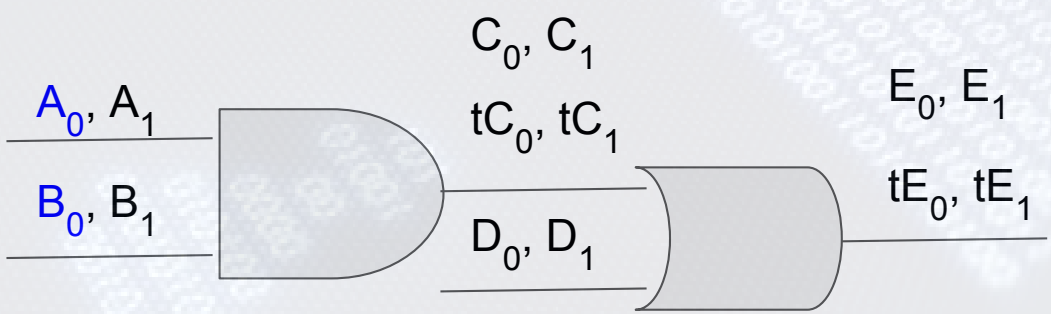
$H(GC) = h$

$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h$ ✔

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



GT1 GT2

$$h = \begin{array}{|l|}\hline E_{A0,B0}(tC_0) \oplus \Delta \\ \hline E_{A0,B1}(tC_0) \\ \hline E_{A1,B0}(tC_0) \\ \hline E_{A1,B1}(tC_0) \\ \hline \end{array} \oplus \begin{array}{|l|}\hline E_{C0,D0}(tE_0) \\ \hline E_{C0,D1}(tE_1) \\ \hline E_{C1,D0}(tE_1) \\ \hline E_{C1,D1}(tE_1) \\ \hline \end{array}$$

$GC = (GT1, GT2)$

$H(GC) = h$

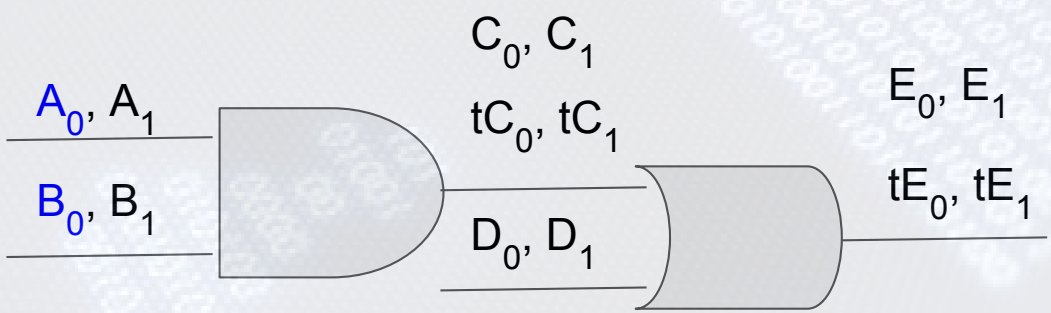
$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h$ ✔

$tC_0 \oplus GT1 = C_0$

$C_b = tC_b \oplus GT1$

$E_b = tE_b \oplus GT2$



GT1 GT2

$$h = \begin{array}{|l|}\hline E_{A0,B0}(tC_0) \oplus \Delta \\ \hline E_{A0,B1}(tC_0) \\ \hline E_{A1,B0}(tC_0) \\ \hline E_{A1,B1}(tC_0) \\ \hline \end{array} \oplus \begin{array}{|l|}\hline E_{C0,D0}(tE_0) \\ \hline E_{C0,D1}(tE_1) \\ \hline E_{C1,D0}(tE_1) \\ \hline E_{C1,D1}(tE_1) \\ \hline \end{array}$$

$GC = (GT1, GT2)$

$H(GC) = h$

$\hat{GC} = (\hat{GT1}, GT2)$

$H(\hat{GC}) = h$ ✔

$tC_0 \oplus GT1 = C_0$ ✔

GC hash construction

- $\mathcal{A}$ modifies a GT entry, and corrects it within the same table

GC hash construction

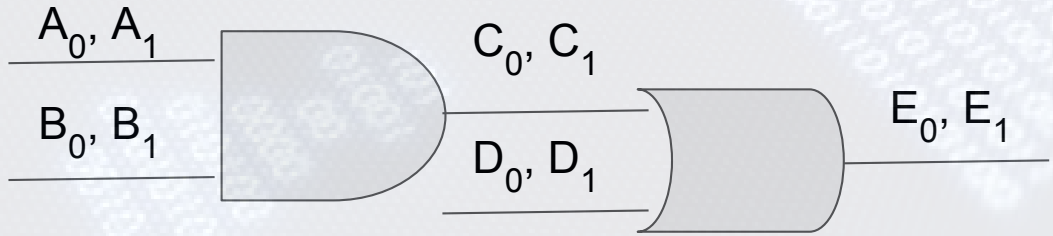
- $\mathcal{A}$ modifies a GT entry, and corrects it within the same table
- Works since the "fix" for broken hash also fixes the translation from temporary to real wire label

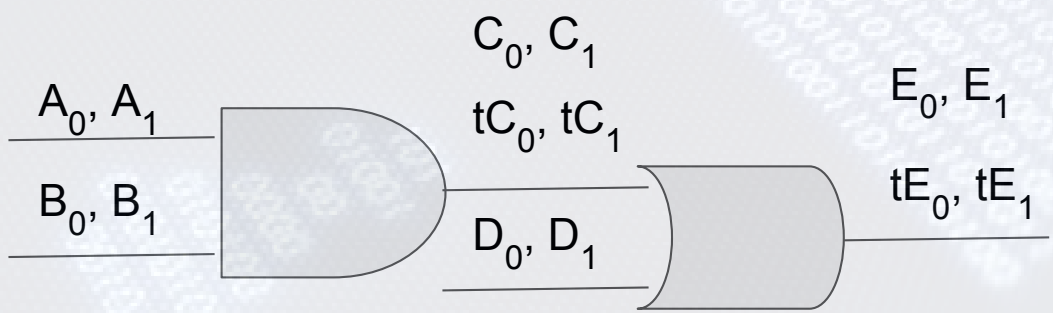
GC hash construction

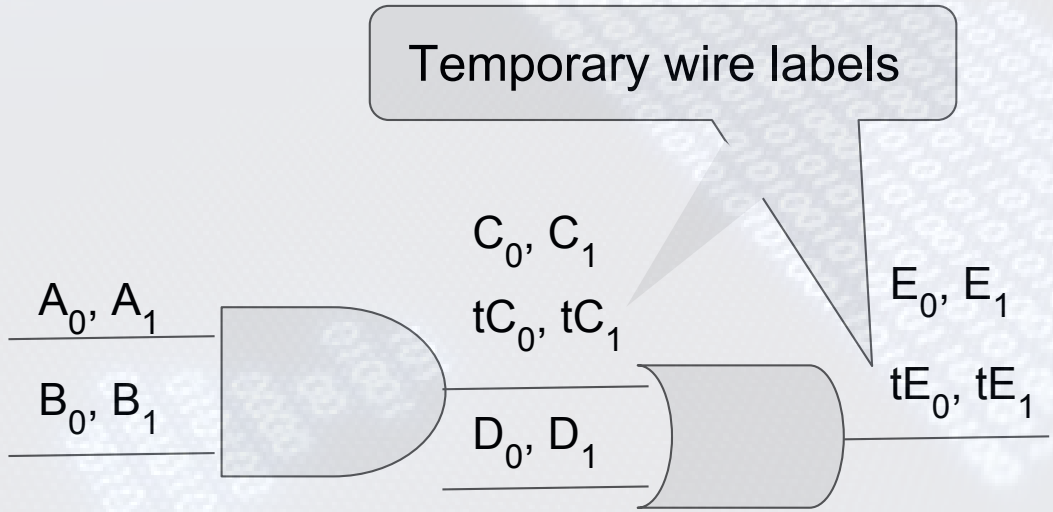
- $\mathcal{A}$ modifies a GT entry, and corrects it within the same table
- Works since the "fix" for broken hash also fixes the translation from temporary to real wire label
- Use GT rows for computing wire label and hash in *different ways*

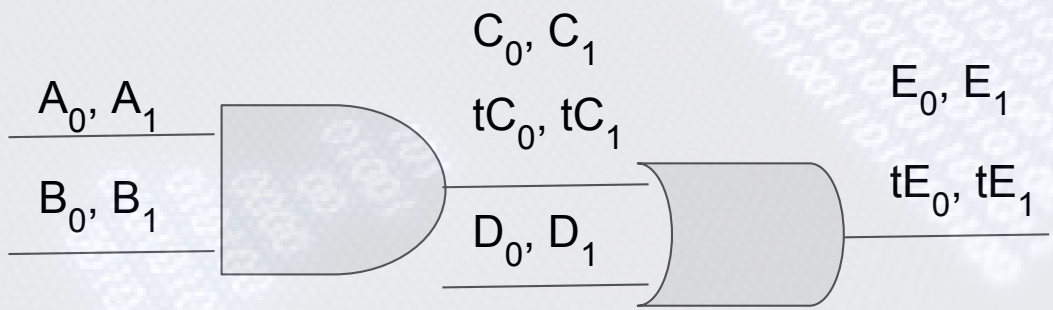
GC hash construction

- $\mathcal{A}$ modifies a GT entry, and corrects it within the same table
- Works since the "fix" for broken hash also fixes the translation from temporary to real wire label
- Use GT rows for computing wire label and hash in *different ways*
- The "fix" for hash will no longer keep the wire label valid


A_0, A_1
B_0, B_1
C_0, C_1
D_0, D_1
E_0, E_1


A_0, A_1
B_0, B_1
C_0, C_1
tC_0, tC_1
D_0, D_1
E_0, E_1
tE_0, tE_1


Temporary wire labels
A_0, A_1
B_0, B_1
C_0, C_1
tC_0, tC_1
D_0, D_1
E_0, E_1
tE_0, tE_1



GT1

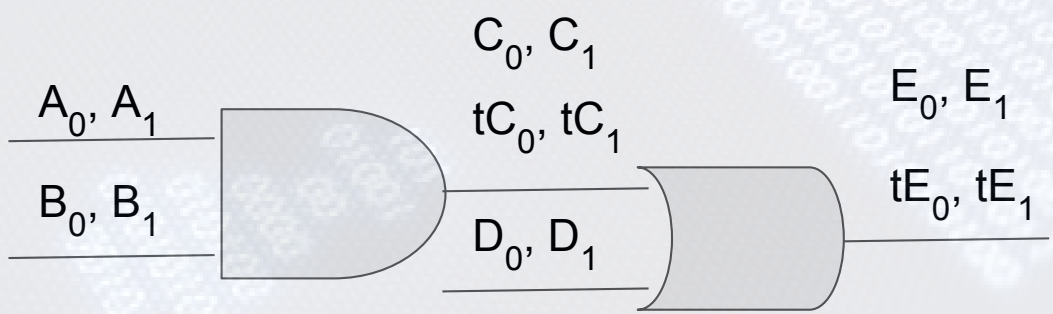
$E_{A0, B0}\ (tC_0)$
$E_{A0, B1}\ (tC_0)$
$E_{A1, B0}\ (tC_0)$
$E_{A1, B1}\ (tC_1)$

GT2

$E_{C0, D0}\ (tE_0)$
$E_{C0, D1}\ (tE_1)$
$E_{C1, D0}\ (tE_1)$
$E_{C1, D1}\ (tE_1)$

$C_b = tC_b \oplus f$ (GT1)

$E_b = tE_b \oplus f$ (GT2)



GT1

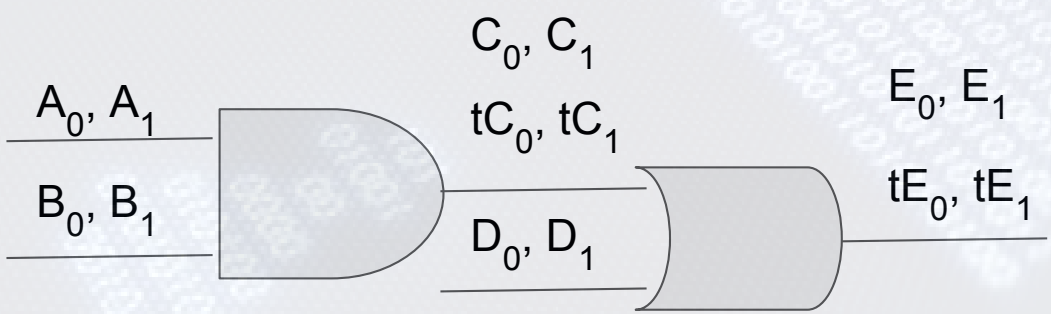
$E_{A0, B0}\ (tC_0)$
$E_{A0, B1}\ (tC_0)$
$E_{A1, B0}\ (tC_0)$
$E_{A1, B1}\ (tC_1)$

GT2

$E_{C0, D0}\ (tE_0)$
$E_{C0, D1}\ (tE_1)$
$E_{C1, D0}\ (tE_1)$
$E_{C1, D1}\ (tE_1)$

$C_b = tC_b \oplus f$ (GT1)

$E_b = tE_b \oplus f$ (GT2)



GT1 GT2

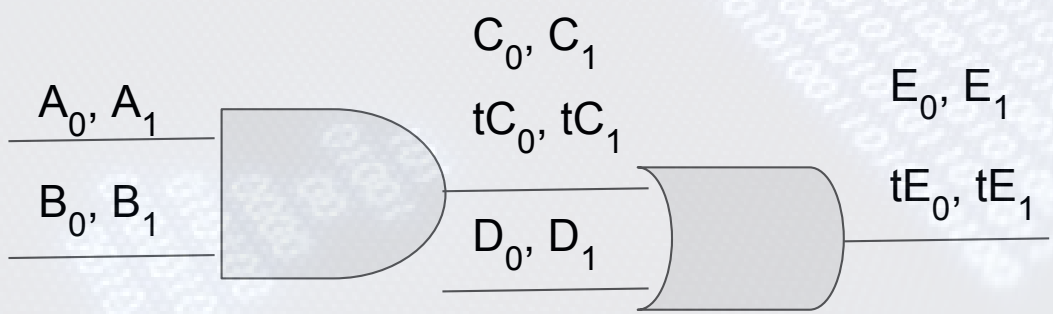
$$h = \begin{array}{|c|}\hline E_{A0,B0}(tC_0) \\ \hline E_{A0,B1}(tC_0) \\ \hline E_{A1,B0}(tC_0) \\ \hline E_{A1,B1}(tC_1) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline E_{C0,D0}(tE_0) \\ \hline E_{C0,D1}(tE_1) \\ \hline E_{C1,D0}(tE_1) \\ \hline E_{C1,D1}(tE_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

$H(GC) = h = GT1 \oplus GT2$

$C_b = tC_b \oplus f$ (GT1)

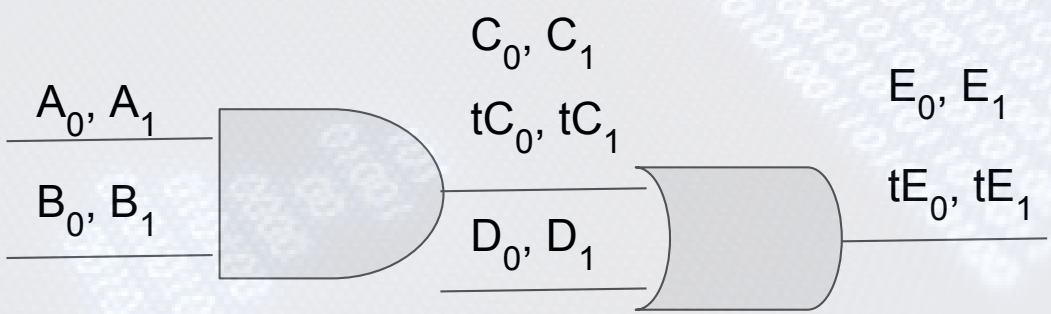
$E_b = tE_b \oplus f$ (GT2)



$$h = \begin{array}{l} R_1 \\ R_2 \\ R_3 \\ R_4 \end{array} \begin{array}{|c|} \hline E_{A0,\,B0}\,(tC_0) \\ \hline E_{A0,\,B1}\,(tC_0) \\ \hline E_{A1,\,B0}\,(tC_0) \\ \hline E_{A1,\,B1}\,(tC_1) \\ \hline \end{array} \oplus \begin{array}{|c|} \hline E_{C0,\,D0}\,(tE_0) \\ \hline E_{C0,\,D1}\,(tE_1) \\ \hline E_{C1,\,D0}\,(tE_1) \\ \hline E_{C1,\,D1}\,(tE_1) \\ \hline \end{array}$$

GC = (GT1, GT2)

H(GC) = h = GT1$\oplus$GT2



$C_b = tC_b \oplus f\ (GT1)$

$E_b = tE_b \oplus f\ (GT2)$

$$h = \begin{matrix} R_1 \\ R_2 \\ R_3 \\ R_4 \end{matrix} \begin{matrix} E_{A0,\,B0}\ (tC_0) \\ E_{A0,\,B1}\ (tC_0) \\ E_{A1,\,B0}\ (tC_0) \\ E_{A1,\,B1}\ (tC_1) \end{matrix} \oplus \begin{matrix} E_{C0,\,D0}\ (tE_0) \\ E_{C0,\,D1}\ (tE_1) \\ E_{C1,\,D0}\ (tE_1) \\ E_{C1,\,D1}\ (tE_1) \end{matrix}$$

$GC = (GT1, GT2)$

$H(GC) = h = GT1 \oplus GT2$

$f\ (GT) = \oplus R_i^{<< i}$

GC hash construction

- Use GT rows as XOR pads in a different manner for computing the GC hash and for offsetting the wire values

GC hash construction

- Use GT rows as XOR pads in a different manner for computing the GC hash and for offsetting the wire values
- Fix for the hash will not simultaneously keep the wire label valid, w.h.p.

GC hash construction

- Use GT rows as XOR pads in a different manner for computing the GC hash and for offsetting the wire values
- Fix for the hash will not simultaneously keep the wire label valid, w.h.p.
- $GC \neq \hat{GC}, H(GC) = H(\hat{GC})$, evaluation of $\hat{GC}$ will fail

GC hash construction

- Bit shifting – fast and easy to implement

GC hash construction

- Bit shifting – fast and easy to implement
- In general, functions f_i such that, if

$$\bigoplus_{i=1}^{4} R_i = \bigoplus_{i=1}^{4} \widehat{R}_i$$

for $R_i \neq \widehat{R}_i$ Then, w.h.p.,

$$\bigoplus_{i=1}^{4} f_i(R_i) \neq \bigoplus_{i=1}^{4} f_i(\widehat{R}_i)$$

GC hash construction

- Bit shifting – fast and easy to implement
- In general, functions f_i such that, if

$$\bigoplus_{i=1}^{4} R_i = \bigoplus_{i=1}^{4} \widehat{R}_i$$

for $R_i \neq \widehat{R}_i$ Then, w.h.p.,

$$\bigoplus_{i=1}^{4} f_i(R_i) \neq \bigoplus_{i=1}^{4} f_i(\widehat{R}_i)$$

- (i.e. if GC is changed s.t. XOR of GT rows is the same, then w.h.p. XOR of f(GT) will change)

Assumptions

- Instantiate key derivation functions

Assumptions

- Instantiate key derivation functions
- $H(X, i) = \pi(K) \oplus K, K = 2x \oplus i$ (π an ideal cipher, instantiated with 128-bit AES with randomly chosen key)

Assumptions

- Instantiate key derivation functions
- $H(X, i) = \pi(K) \oplus K, K = 2x \oplus i$ (π an ideal cipher, instantiated with 128-bit AES with randomly chosen key)
- Davies-Meyer meets the guarantees of the random permutation model

Assumptions

- Instantiate key derivation functions
- $H(X, i) = \pi(K) \oplus K, K = 2x \oplus i$ (π an ideal cipher, instantiated with 128-bit AES with randomly chosen key)
- Davies-Meyer meets the guarantees of the random permutation model
- Free-XOR – DM is correlation-robust

Assumptions

- Instantiate key derivation functions
- $H(X, i) = \pi(K) \oplus K, K = 2x \oplus i$ (π an ideal cipher, instantiated with 128-bit AES with randomly chosen key)
- Davies-Meyer meets the guarantees of the random permutation model
- Free-XOR – DM is correlation-robust
- Hash security

Assumptions

- Instantiate key derivation functions
- $H(X, i) = \pi(K) \oplus K, K = 2x \oplus i$ (π an ideal cipher, instantiated with 128-bit AES with randomly chosen key)
- Davies-Meyer meets the guarantees of the random permutation model
- Free-XOR – DM is correlation-robust
- Hash security
  - Collision resistance of DM

Assumptions

- Instantiate key derivation functions
- $H(X, i) = \pi(K) \oplus K, K = 2x \oplus i$ (π an ideal cipher, instantiated with 128-bit AES with randomly chosen key)
- Davies-Meyer meets the guarantees of the random permutation model
- Free-XOR – DM is correlation-robust
- Hash security
  - Collision resistance of DM
  - Can be achieved assuming DM is an ideal cipher

Half-gate garbling

- ZRE'15 – state-of-the-art in garbling

Half-gate garbling

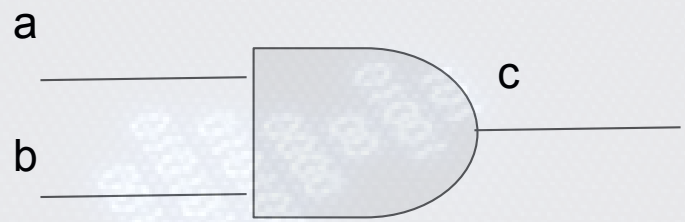
- ZRE'15 – state-of-the-art in garbling
- Compatible with free-XOR

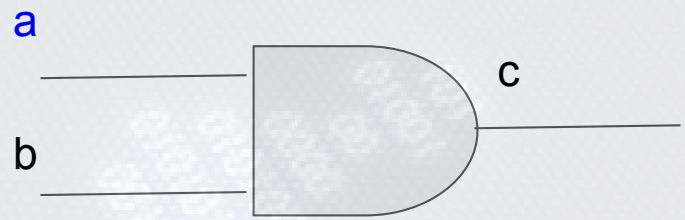
Half-gate garbling

- ZRE'15 – state-of-the-art in garbling
- Compatible with free-XOR
- 2 ciphertexts for AND gate

Half-gate garbling

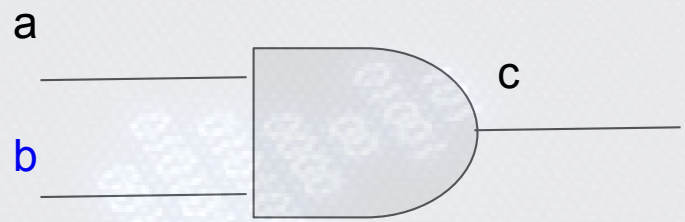
- ZRE'15 – state-of-the-art in garbling
- Compatible with free-XOR
- 2 ciphertexts for AND gate
- Free hash for half-gates garbling?


a
b
c



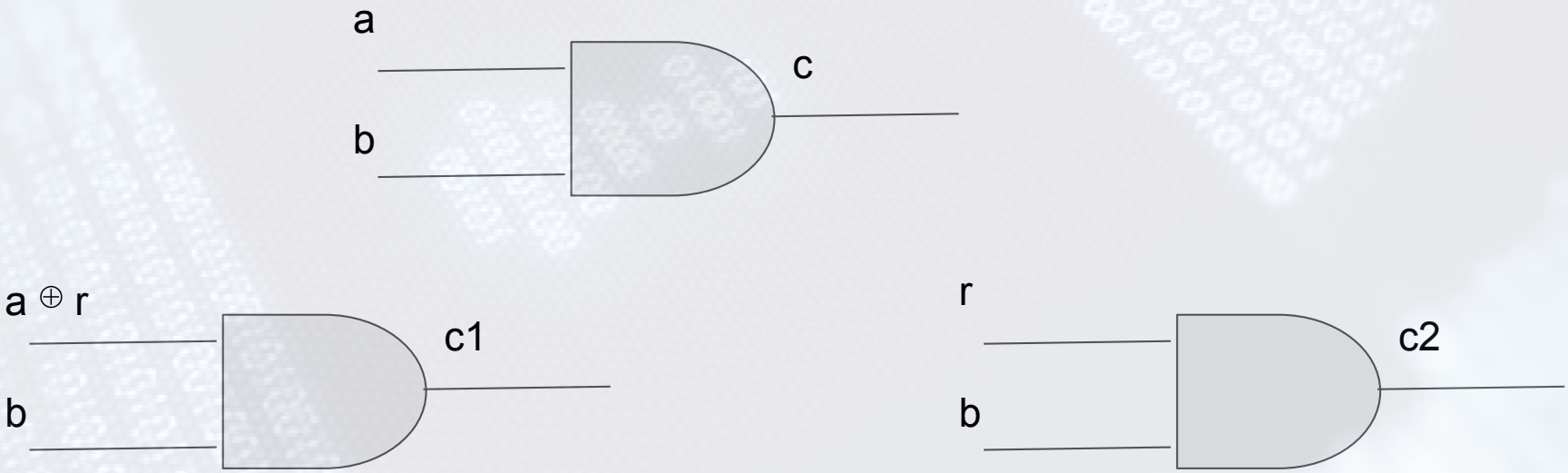
Garbler knows one of the values in the clear

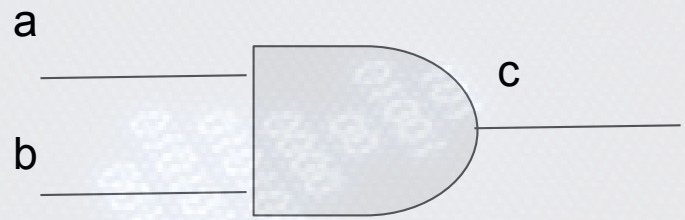
Generator half-gate

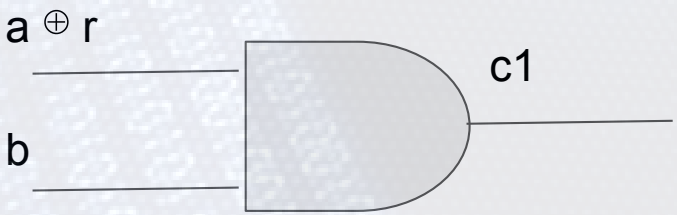


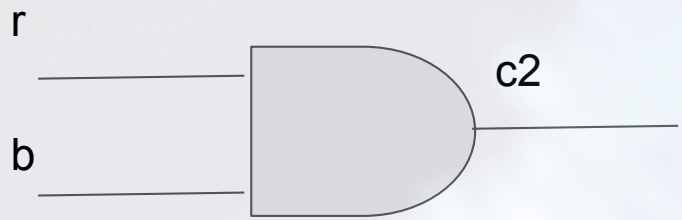
Evaluator knows one of the values in the clear

Evaluator half-gate

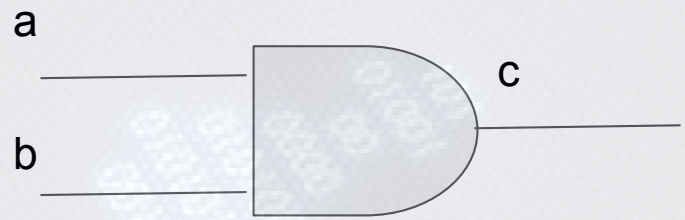

a
b
c
a ⊕ r
b
c1
r
b
c2

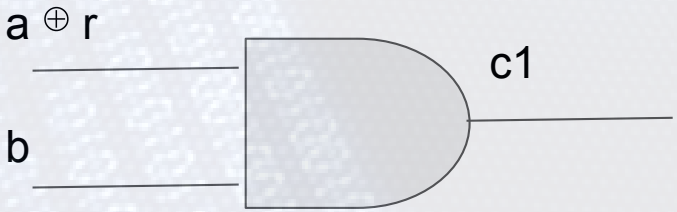


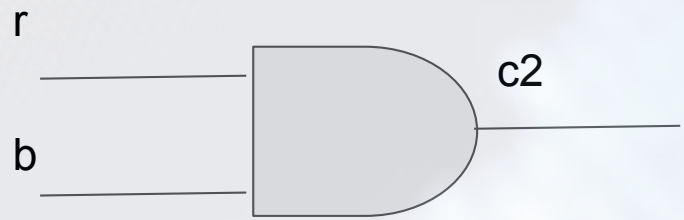




r chosen by the garbler
Evaluator learns a ⊕ r in the clear

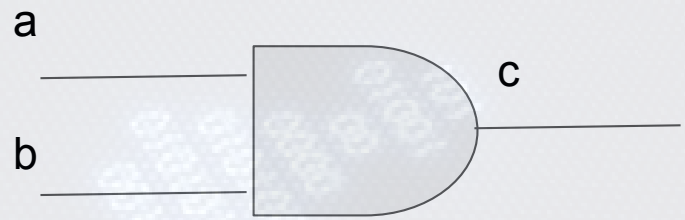


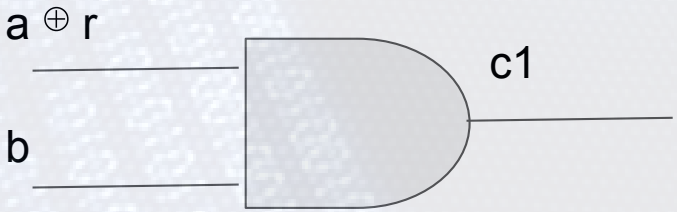


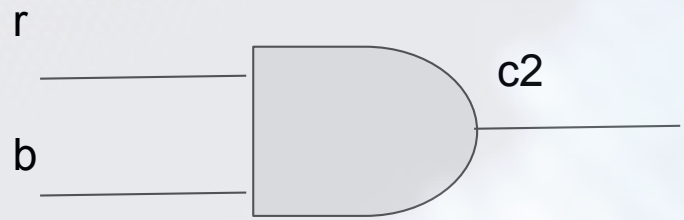


Evaluator half-gate

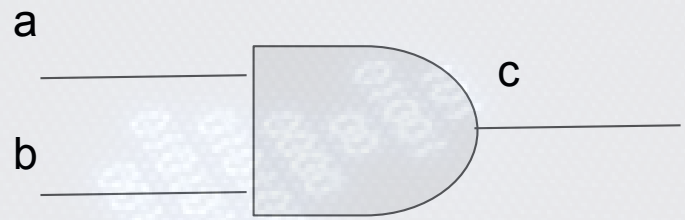
Generator half-gate

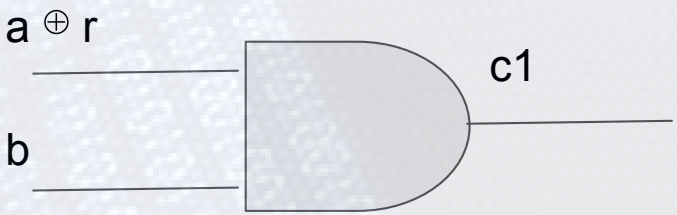


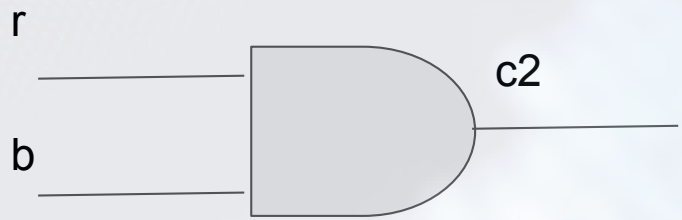




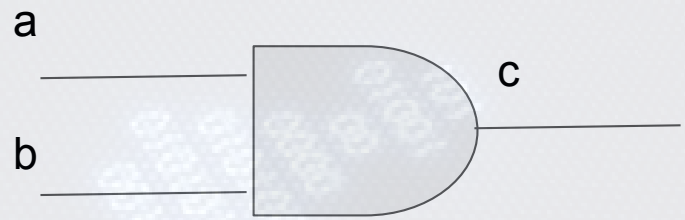
$c1 \oplus c2$

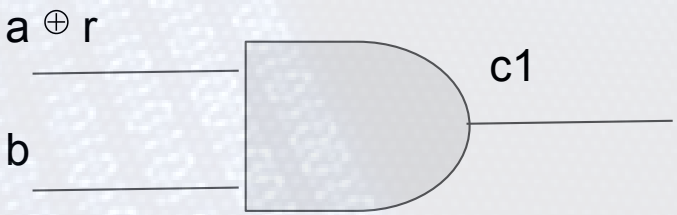


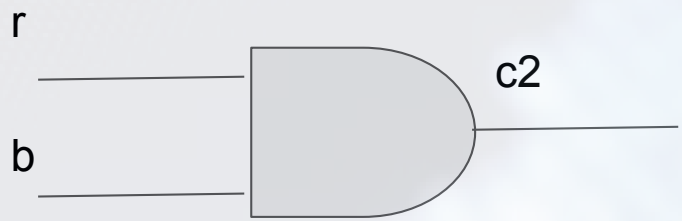




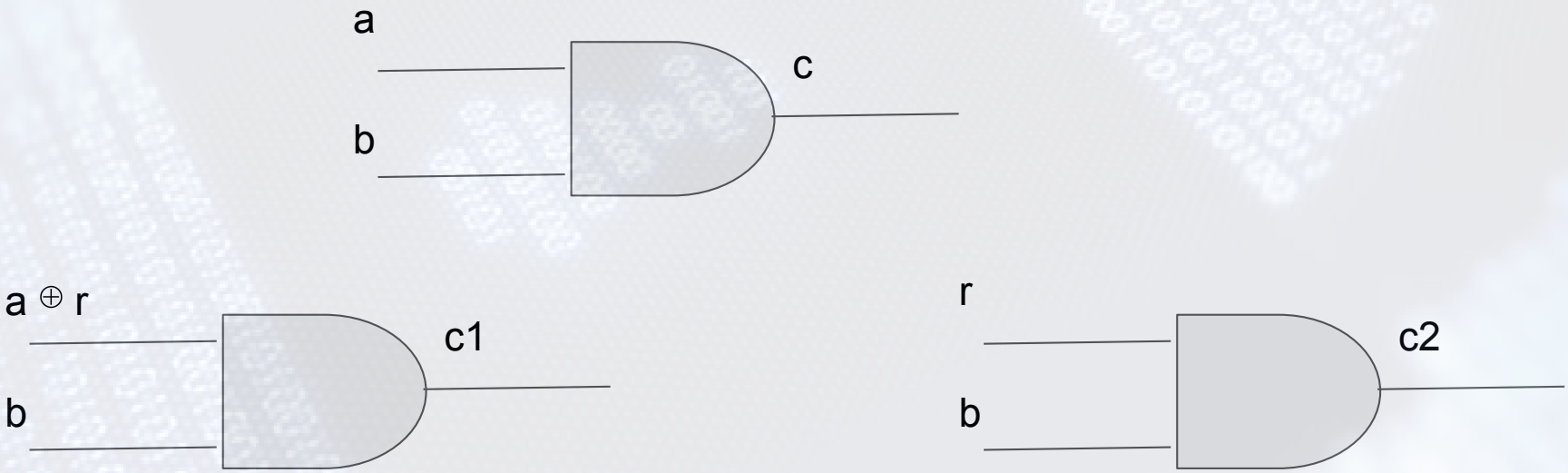
$$c1 \oplus c2 = ((a \oplus r) \wedge b) \oplus (r \wedge b)$$

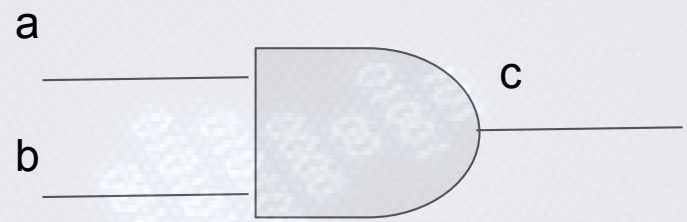


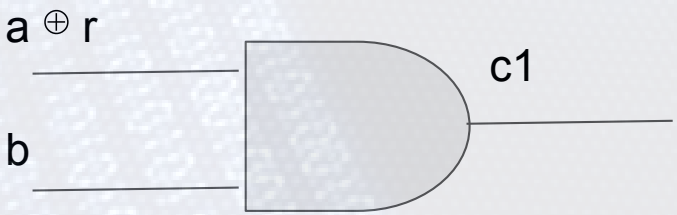


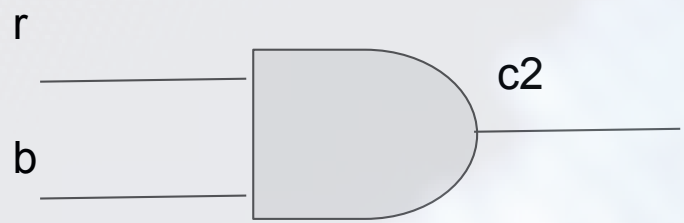


$$c1 \oplus c2 = ((a \oplus r) \wedge b) \oplus (r \wedge b) = a \wedge b$$


a
b
c
a ⊕ r
b
c1
r
b
c2

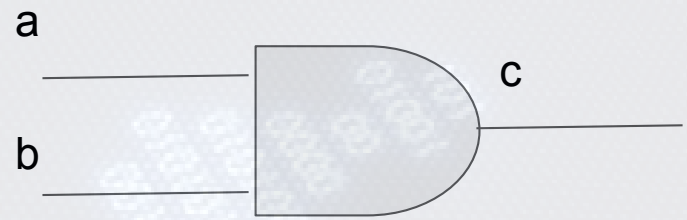






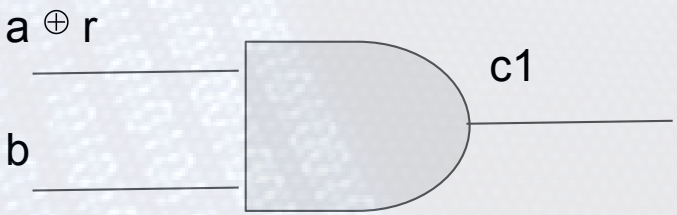
$E_{B \oplus R}(C1)$

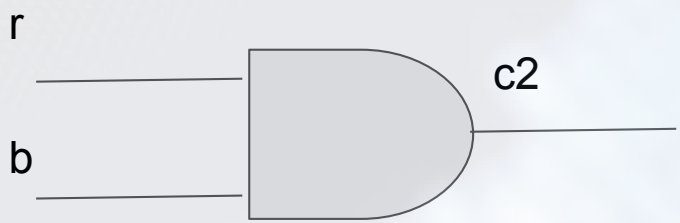
$E_{B \oplus R}(C2)$



$E_{B \oplus R}(C1)$

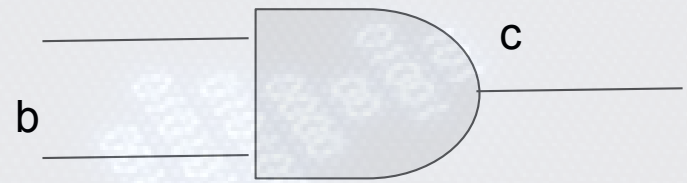
$E_{B \oplus R}(C2)$





$E_{B \oplus R}(C1)$

$E_{B \oplus R}(C2)$

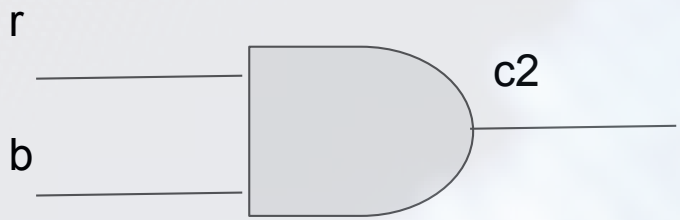


$E_{B \oplus R}(C1)$

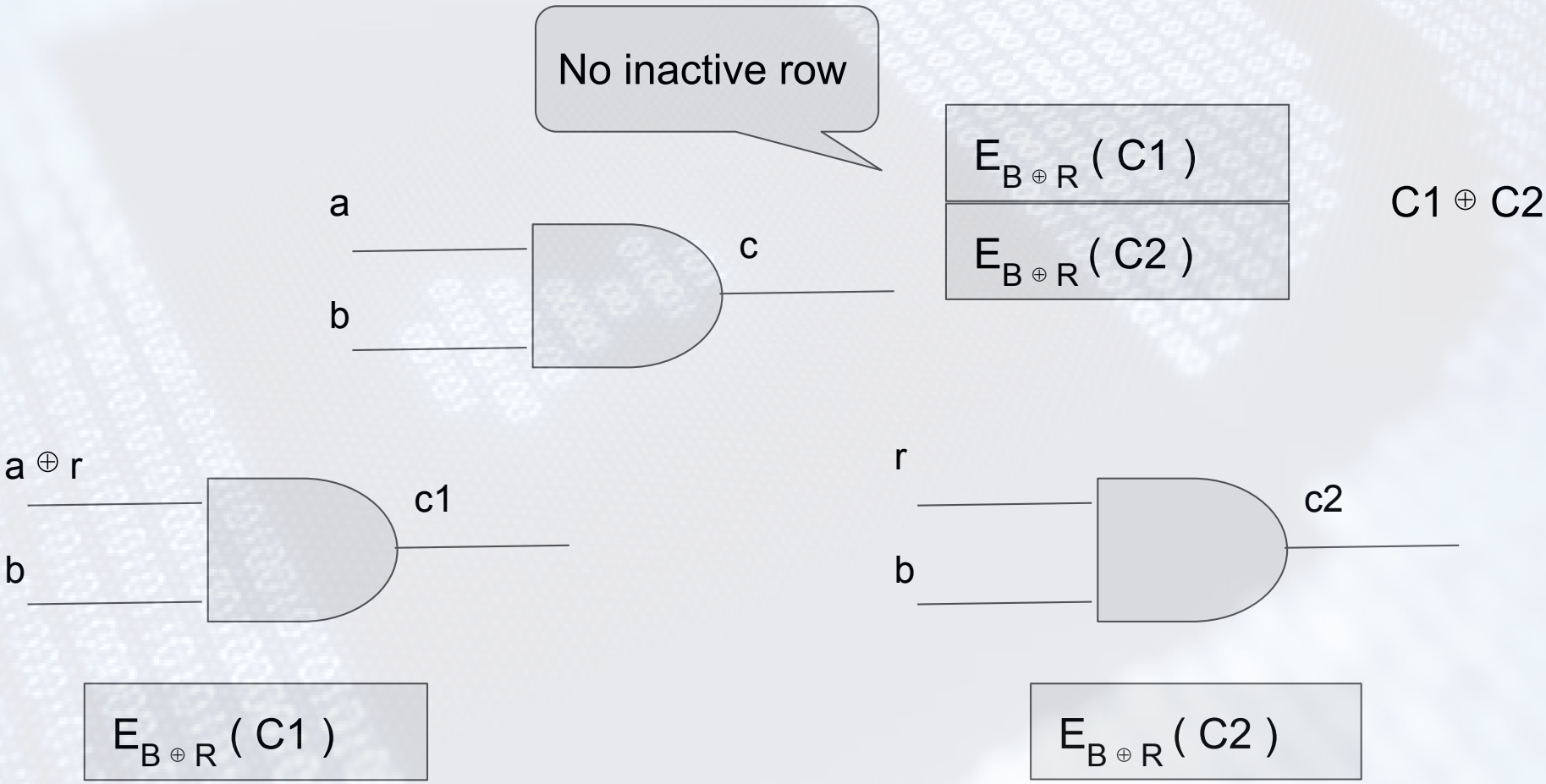
$E_{B \oplus R}(C2)$

$C1 \oplus C2$

$E_{B \oplus R}(C1)$



$E_{B \oplus R}(C2)$


No inactive row
$E_{B \oplus R}(C1)$
$E_{B \oplus R}(C2)$
$C1 \oplus C2$
a
b
c
$a \oplus r$
b
c1
$E_{B \oplus R}(C1)$
r
b
c2
$E_{B \oplus R}(C2)$

Hashing in half-gate garbling

- Observation – Both ciphertexts decrypted and used to compute output wire label

Hashing in half-gate garbling

- Observation – Both ciphertexts decrypted and used to compute output wire label
- Modifying a garbled row causes unpredictable change in output wire label

Hashing in half-gate garbling

- Observation – Both ciphertexts decrypted and used to compute output wire label
- Modifying a garbled row causes unpredictable change in output wire label
- Simpler hash construction

Hashing in half-gate garbling

- Observation – Both ciphertexts decrypted and used to compute output wire label
- Modifying a garbled row causes unpredictable change in output wire label
- Simpler hash construction
- $h = H(GC) =$ XOR of all ciphertexts

Overview

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation
- Impact – Applications of free hash

Overview

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation
- Impact – Applications of free hash

Implementation

	Our construction	Garble + SHA	justGarble
Standard Garbling	31.1	226.7	29
Half-gates	26.8	157.7	25.3

- AES circuit garbled, numbers in cycles per gate
- libgarble, AES-NI integrated
- The configuration: 2.3 GHz Core i5-2410M processor with 4 GB RAM

Implementation

	Our construction	Garble + SHA	justGarble
Standard Garbling	31.1	226.7	29
Half-gates	26.8	157.7	25.3

- AES circuit garbled, numbers in cycles per gate
- libgarble, AES-NI integrated
- The configuration: 2.3 GHz Core i5-2410M processor with 4 GB RAM

Overview

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation
- Impact – Applications of free hash

Overview

- Definition of GC hash security
- Hashed garbling constructions – standard garbling and half-gates [ZRE'15]
- Implementation and evaluation
- Impact – Applications of free hash

CR hash vs free hash

- CR hash

CR hash vs free hash

- CR hash
  - P_1 commits to GC GC via $h = H(GC)$

CR hash vs free hash

- CR hash
  - P_1 commits to GC GC via $h = H(GC)$
  - GC can be good or cheating

CR hash vs free hash

- CR hash
  - P_1 commits to GC GC via $h = H(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation

CR hash vs free hash

- CR hash
  - P_1 commits to GC GC via $h = H(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation
- Free hash

CR hash vs free hash

- CR hash
  - P_1 commits to GC GC via $h = H(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation
- Free hash
  - P_1 commits to GC GC via $h = hG(GC)$

CR hash vs free hash

- CR hash
  - P_1 commits to GC GC via $h = H(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation
- Free hash
  - P_1 commits to GC GC via $h = hG(GC)$
  - GC can be good or cheating

CR hash vs free hash

- CR hash
  - P_1 commits to GC GC via $h = H(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation
- Free hash
  - P_1 commits to GC GC via $h = hG(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation

CR hash vs free hash

- CR hash
  - P_1 commits to GC GC via $h = H(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation
- Free hash
  - P_1 commits to GC GC via $h = hG(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation
  - P_1 can open {good,cheating} $\rightarrow$ broken (fail evaluation)

CR hash vs free hash

- CR hash
  - P_1 commits to GC GC via $h = H(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation
- Free hash
  - P_1 commits to GC GC via $h = hG(GC)$
  - GC can be good or cheating
  - Once h fixed, P_1 cannot change cheating/good designation
  - P_1 can open {good,cheating} $\rightarrow$ broken (fail evaluation)
  - When can P_2 abort? (cf. selective failure)

Covert secure protocols

- Covert model – a party can deviate from the protocol, but is caught with a fixed probability, the deterrence factor

Covert secure protocols

- Covert model – a party can deviate from the protocol, but is caught with a fixed probability, the deterrence factor
- Introduced in AL'07, public verifiability (PVC) studied in AO'12, KM'15

Covert secure protocols

- Covert model – a party can deviate from the protocol, but is caught with a fixed probability, the deterrence factor
- Introduced in AL'07, public verifiability (PVC) studied in AO'12, KM'15
- Cheating P_1 can turn a good evaluation circuit into a broken one

Covert secure protocols

- Covert model – a party can deviate from the protocol, but is caught with a fixed probability, the deterrence factor
- Introduced in AL'07, public verifiability (PVC) studied in AO'12, KM'15
- Cheating P_1 can turn a good evaluation circuit into a broken one
- P_2 can safely abort – independent of input

Covert secure protocols

- Covert model – a party can deviate from the protocol, but is caught with a fixed probability, the deterrence factor
- Introduced in AL'07, public verifiability (PVC) studied in AO'12, KM'15
- Cheating P_1 can turn a good evaluation circuit into a broken one
- P_2 can safely abort – independent of input
- Deterrence improvement for the same communication complexity

Covert secure protocols

- Covert model – a party can deviate from the protocol, but is caught with a fixed probability, the deterrence factor
- Introduced in AL'07, public verifiability (PVC) studied in AO'12, KM'15
- Cheating P_1 can turn a good evaluation circuit into a broken one
- P_2 can safely abort – independent of input
- Deterrence improvement for the same communication complexity
- Total execution time improved for the same deterrence

Covert secure protocols – improving performance

	Total number of circuits	Number of check circuits	Circuits sent**	Time (in secs)
AL'07	10	9	10	3510
AL'07+free hash	10	9	1	1260
KM'15	10	9	10	3510
KM'15+free hash	10	9	1	1260

- Execution time estimates with deterrence of $\epsilon = 0.9$.

Covert secure protocols – improving performance

	Total number of circuits	Number of check circuits	Circuits sent**	Time (in secs)
AL'07	10	9	10	3510
AL'07+free hash	10	9	1	1260
KM'15	10	9	10	3510
KM'15+free hash	10	9	1	1260

- Execution time estimates with deterrence of $\epsilon = 0.9$.
- GC generation for a circuit with 1 billion gates – 95 seconds (per JustGarble paper).

Covert secure protocols – improving performance

	Total number of circuits	Number of check circuits	Circuits sent**	Time (in secs)
AL'07	10	9	10	3510
AL'07+free hash	10	9	1	1260
KM'15	10	9	10	3510
KM'15+free hash	10	9	1	1260

- Execution time estimates with deterrence of $\epsilon = 0.9$.
- GC generation for a circuit with 1 billion gates – 95 seconds (per JustGarble paper).
- Communication: assuming 1Gbps channel – send 1 billion bits/sec.

Covert secure protocols – improving performance

	Total number of circuits	Number of check circuits	Circuits sent**	Time (in secs)
AL'07	10	9	10	3510
AL'07+free hash	10	9	1	1260
KM'15	10	9	10	3510
KM'15+free hash	10	9	1	1260

- Execution time estimates with deterrence of $\epsilon = 0.9$.
- GC generation for a circuit with 1 billion gates – 95 seconds (per JustGarble paper).
- Communication: assuming 1Gbps channel – send 1 billion bits/sec.
- Time to send a circuit of 1 billion gates is 256 seconds (assuming half gates and 2×128 bits per gate).

Covert secure protocols – improving performance

	Total number of circuits	Number of check circuits	Circuits sent**	Time (in secs)
AL'07	10	9	10	3510
AL'07+free hash	10	9	1	1260
KM'15	10	9	10	3510
KM'15+free hash	10	9	1	1260

- Execution time estimates with deterrence of $\epsilon = 0.9$.
- GC generation for a circuit with 1 billion gates – 95 seconds (per JustGarble paper).
- Communication: assuming 1Gbps channel – send 1 billion bits/sec.
- Time to send a circuit of 1 billion gates is 256 seconds (assuming half gates and 2×128 bits per gate).
- ** In AL07 and KM15 cheaper to send GC than to compute SHA (GC)

Other applications

- Apply to maliciously secure protocols?

Other applications

- Apply to maliciously secure protocols?
- Does not apply to state-of-the-art protocol of Lin'13

Other applications

- Apply to maliciously secure protocols?
- Does not apply to state-of-the-art protocol of Lin'13
  - Cheating punishment relies on at least one evaluation circuit being "good"

Other applications

- Apply to maliciously secure protocols?
- Does not apply to state-of-the-art protocol of Lin'13
  - Cheating punishment relies on at least one evaluation circuit being "good"
  - P_1 can open the good evaluation circuit as a "broken" one

Other applications

- Apply to maliciously secure protocols?
- Does not apply to state-of-the-art protocol of Lin'13
  - Cheating punishment relies on at least one evaluation circuit being "good"
  - P_1 can open the good evaluation circuit as a "broken" one
  - P_2 *cannot tell* if decoding failure due to selective failure or hash failure

Other applications

- Apply to maliciously secure protocols?
- Does not apply to state-of-the-art protocol of Lin'13
  - Cheating punishment relies on at least one evaluation circuit being "good"
  - P_1 can open the good evaluation circuit as a "broken" one
  - P_2 *cannot tell* if decoding failure due to selective failure or hash failure
  - This prevents input recovery

Other applications

- Apply to maliciously secure protocols?
- Does not apply to state-of-the-art protocol of Lin'13
  - Cheating punishment relies on at least one evaluation circuit being "good"
  - P_1 can open the good evaluation circuit as a "broken" one
  - P_2 *cannot tell* if decoding failure due to selective failure or hash failure
  - This prevents input recovery
- Does not apply to dual-execution protocols (HKE'13, KMR'15)

Other applications

- Apply to maliciously secure protocols?
- Does not apply to state-of-the-art protocol of Lin'13
  - Cheating punishment relies on at least one evaluation circuit being "good"
  - P_1 can open the good evaluation circuit as a "broken" one
  - P_2 *cannot tell* if decoding failure due to selective failure or hash failure
  - This prevents input recovery
- Does not apply to dual-execution protocols (HKE'13, KMR'15)
  - Open all "good" evaluation circuits as "broken" ones

Other applications

- Apply to maliciously secure protocols?
- Does not apply to state-of-the-art protocol of Lin'13
  - Cheating punishment relies on at least one evaluation circuit being "good"
  - P_1 can open the good evaluation circuit as a "broken" one
  - P_2 *cannot tell* if decoding failure due to selective failure or hash failure
  - This prevents input recovery
- Does not apply to dual-execution protocols (HKE'13, KMR'15)
  - Open all "good" evaluation circuits as "broken" ones
  - PSI output leaks information

What else?

- Can be used in majority-based protocols

What else?

- Can be used in majority-based protocols
- Calculation of optimal ratio when check circuits are cheaper – in the paper

What else?

- Can be used in majority-based protocols
- Calculation of optimal ratio when check circuits are cheaper – in the paper
- Applicability in amortized settings?

What else?

- Can be used in majority-based protocols
- Calculation of optimal ratio when check circuits are cheaper – in the paper
- Applicability in amortized settings?
- Hash security when $\mathcal{A}$ changes the topology of the circuit?

Thank you!

Thank You!