Yao's Garbled Circuits

Recent Directions and Implementations

Pete Snyder

Outline

1. Context
2. Security definitions
3. Oblivious transfer
4. Yao’s original protocol
5. Security improvements
6. Performance improvements
7. Implementations
8. Conclusion

Outline

1. **Context**
2. Security definitions
3. Oblivious transfer
4. Yao's original protocol
5. Security improvements
6. Performance improvements
7. Implementations
8. Conclusion

1. Context for Yao’s Protocol

- Secure function evaluation
- Computing functions with hidden inputs
- “Millionaires’ problem”

Yao and SFE

- Initially only considered theoretically interesting
- Later became focus of practical work
- Yao never published protocol

Outline

1. Context
2. **Security definitions**
3. Oblivious transfer
4. Yao’s original protocol
5. Security improvements
6. Performance improvements
7. Implementations
8. Conclusion

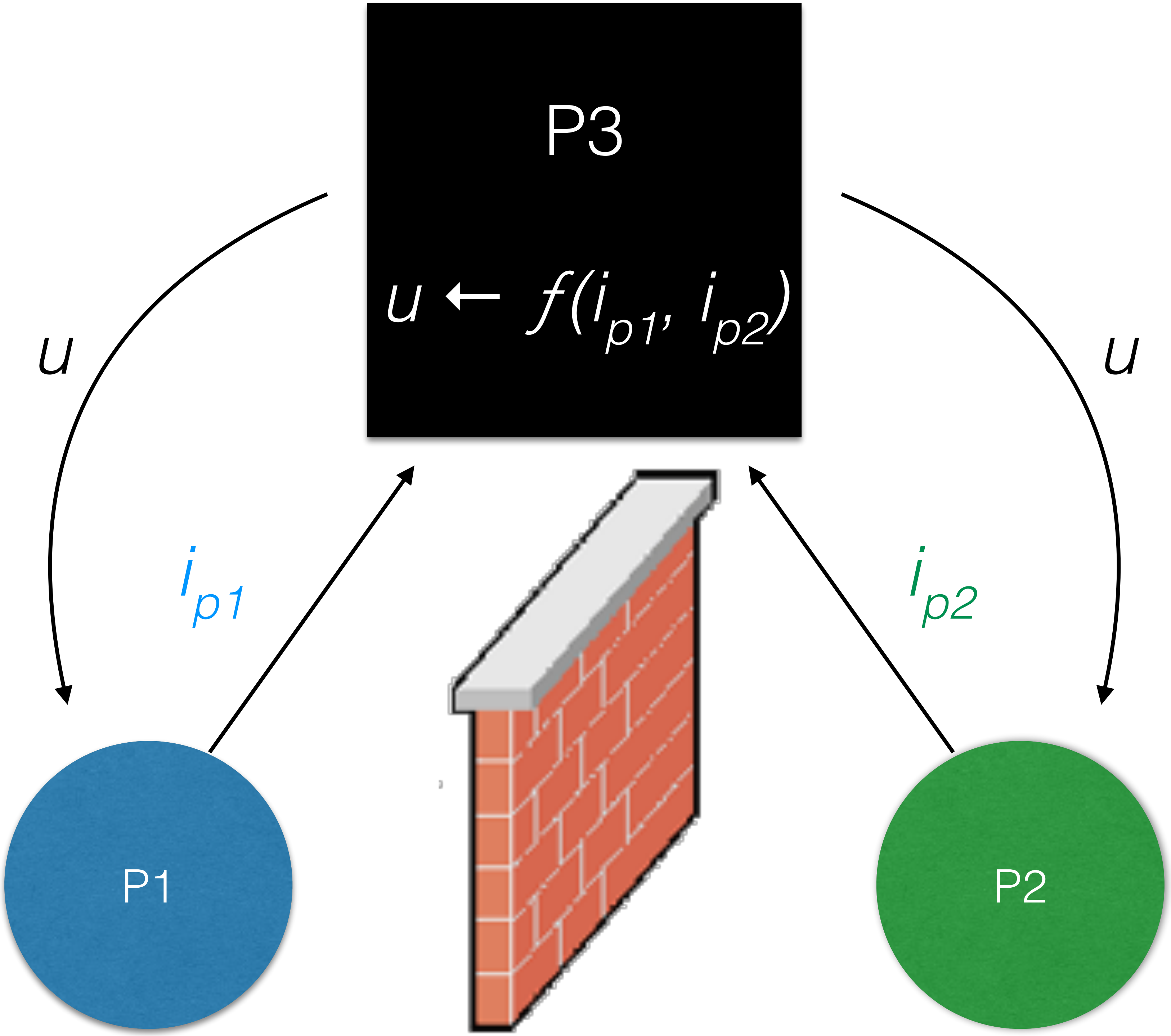
2. Definitions and Assumptions

- Properties of a “secure” SFE protocol
- Adversary models

2.1. SFE Properties

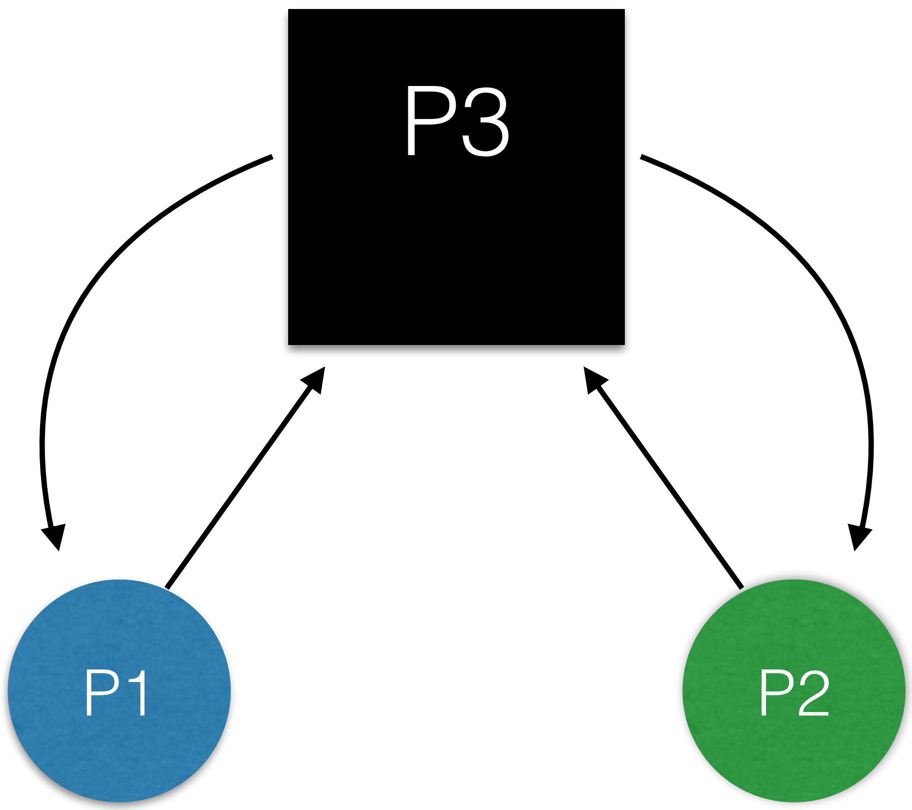
- Could try to fully define what a SFE system can and cannot leak
    - Might quickly devolve into long arbitrary lists
- Instead, compare a solution to a best-possible 3rd party / ideal - oracle

Ideal Oracle



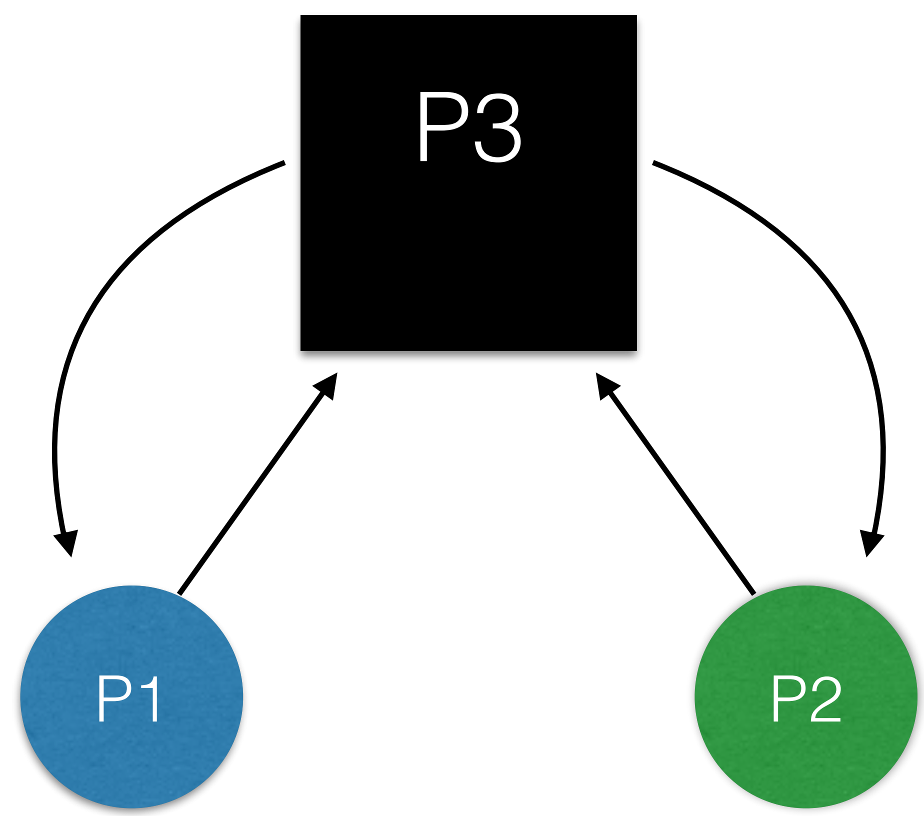
Validity

- A SFE protocol must provide the same result as an ideal oracle
- Does not require:
  - correct answer
  - any answer at all



Privacy

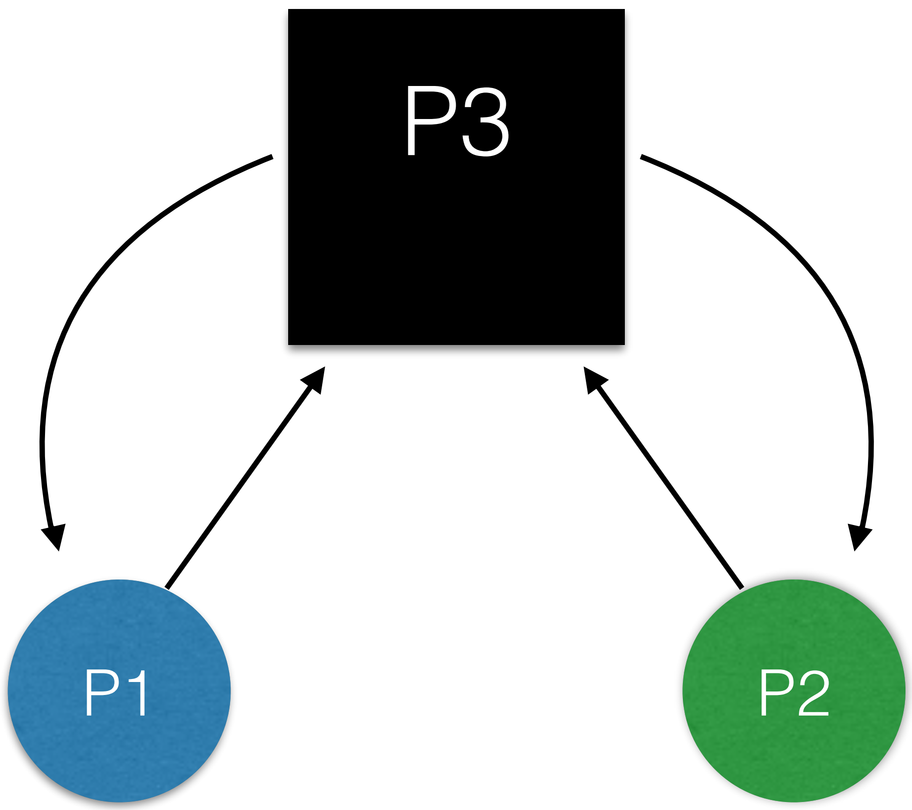
- A SFE protocol must not allow parties to learn more about each other's inputs than they would with an ideal oracle
- Does not require:
  - That parties cannot learn inputs
  - ex: integer multiplication



Fairness

- A SFE protocol must not allow one party to learn result while keeping it from the other.

- Tricky…



2.2. Adversary Models

<u>Semi-Honest</u>

- Follows protocol
- Will take advantage where allowed
- Has transcript of entire protocol

<u>Malicious</u>

- Arbitrarily deviates from protocol
- Will take any beneficial actions
- More “real-world”

Outline

1. Context
2. Security definitions
3. **Oblivious transfer**
4. Yao’s original protocol
5. Security improvements
6. Performance improvements
7. Implementations
8. Conclusion

3. Oblivious transfer

- What is oblivious transfer
- Simple protocol

What is Oblivious Transfer

- OTs is category of 2-party protocols
  - P1 has some values
  - P2 learns some values but not others
  - P1 doesn’t know what P2 learns
- Yao’s protocol builds on OT

1-out-of-2 Oblivious Transfer

Inputs

- **P1:** $S = \{s_0, s_1\}$
- **P2:** $i \in \{0, 1\}$

Receives

- **P1:** Nothing
- **P2:** S_i but not S_{i-1}

Example OT Protocol

P1 $\longrightarrow$ $S = \{s_0, s_1\}$

P2 $\longrightarrow$ $i \in \{0, 1\}$

P2 $\longrightarrow$ $(k^{pub}, k^{prv}), (k^{\perp}, \perp)$

P2 $\longrightarrow$ $k_i^{pub} = k^{pub}, k_{i-1}^{pub} = k^{\perp}$

P1 $\longrightarrow$ $c_i = E_{k_i^{pub}}(s_i), c_{i-1} = E_{k_{i-1}^{pub}}(s_{i-1})$

P2 $\longrightarrow$ $s_i = D_{k^{pri}}(c_i), \perp = D_{\perp}(c_{i-1})$

Outline

1. Context
2. Definitions and assumptions
3. Oblivious transfer
4. **Yao's original protocol**
5. Security improvements
6. Performance improvements
7. Implementations
8. Conclusion

4. Yao’s Protocol

- “Intuitive” description (hopefully…)
- Detailed description

Yao’s Garbled Circuits

1. **P1** and **P2** want to securely compute f
2. **P1**: Creates circuit representation of f
3. **P1**: “garbles” the circuit so that **P2** can execute the circuit, but not learn intermediate values
4. **P1**: Sends **P2** the garbled circuit and his garbled input bits
5. **P2**: Uses OT to receive **P2**’s input bits
6. **P2**: Evaluates circuit

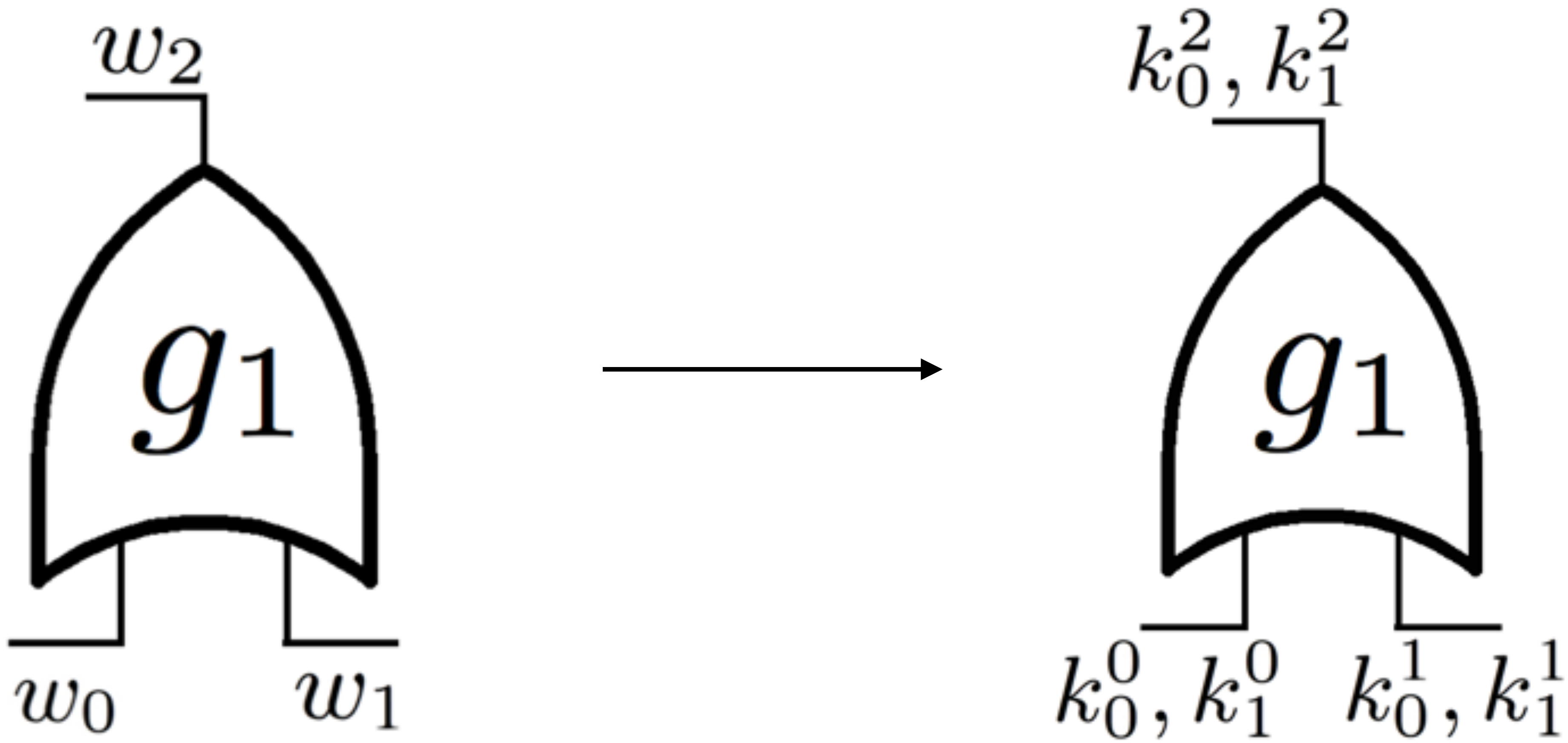
1. Generating equivalent boolean circuit for the function

- Create circuit *c* such that $\forall$x, y -> f(x, y) = c(x, y)
- Beyond this talk (compiler theory, etc.)
- Implementations use domain specific high level languages

2. Garbling the circuit

- Goal is to allow **P2** to compute circuit w/o knowing intermediate values of circuit
- Garbling means mapping binary values to encryption keys, and encrypting outputs of gates
- Pre-garbling: Gates are $\{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$
- Post-garbling: $f(\{0, 1\}^{|k|}, \{0, 1\}^{|k|}) \rightarrow \{0, 1\}^{|k|}$

Preparing one gate



w_0	w_1	w_2
0	0	0
0	1	1
1	0	1
1	1	1

w_0	w_1	w_2	garbled value
k_0^0	k_1^0	k_2^0	$E_{k_0^0}(E_{k_1^0}(k_2^0))$
k_0^0	k_1^1	k_2^1	$E_{k_0^0}(E_{k_1^1}(k_2^1))$
k_0^1	k_1^0	k_2^1	$E_{k_0^1}(E_{k_1^0}(k_2^1))$
k_0^1	k_1^1	k_2^1	$E_{k_0^1}(E_{k_1^1}(k_2^1))$

3. Garbling P1's Input

- **P1** has garbled circuit
- **P1** has original i_{p1}
- **P2** has original i_{p2}
- Circuit only contains garbled / mapped values

Garbling i_{p1}

Original i_{p1}

w	0
w	1
w	1
w	0

Circuit Lookup $\longrightarrow$

Garbled i_{p1}

w	k^0
w	k^1
w	k^1
w	k^0

4. Garbling P2's input

- **P2** has garbled circuit, garbled i_{p1}, original i_{p2}
- **P1** has mappings boolean → garbled mappings
- To compute circuit, **P2** needs garbled input values

Garbling i_{p2}

P1

	0	1
w	k^0	k^1
w	k^0	k^1
w	k^0	k^1
w	k^0	k^1

P2

	i	garbled
w	0	?
w	0	?
w	1	?
w	0	?

Garbling i_{p2}

P1

1-out-of-2 OT

P2

	0	**1**
w	k^0	k^1
w	k^0	k^1
w	k^0	k^1
w	k^0	k^1

$N = \{k_2^0, k_2^1\}$ $i = 0$

	i	**garbled**
w	0	?
w	0	?
w	1	?
w	0	?

Garbling i_{p2}

P1

1-out-of-2 OT

P2

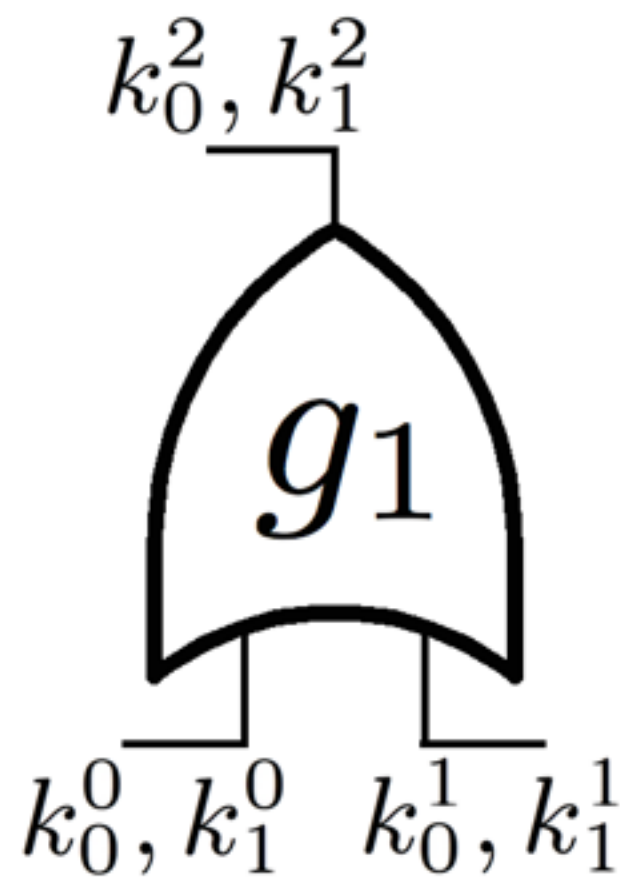
	0	1
w	k^0	k^1
w	k^0	k^1
w	k^0	k^1
w	k^0	k^1

k_2^0

	i	**garbled**
w	0	k^0
w	0	?
w	1	?
w	0	?

5. Computing the circuit

- **P2**: Garbled circuit, i_{p1}, i_{p2}
- **P2**: Tries each row in table to see what key the inputs unlock



Assume **P1**'s input is 1 and **P2**'s input is 0

	garbled value	
$H(k_0^{b_{p1}}\|\|k_0^{b_{p2}}\|\|g_1)\oplus$	$H(k_0^0\|\|k_1^0\|\|g_1) \oplus k_2^0$	$\rightarrow \perp$
$H(k_0^{b_{p1}}\|\|k_0^{b_{p2}}\|\|g_1)\oplus$	$H(k_0^0\|\|k_1^1\|\|g_1) \oplus k_2^1$	$\rightarrow \perp$
$H(k_0^{b_{p1}}\|\|k_0^{b_{p2}}\|\|g_1)\oplus$	$H(k_0^1\|\|k_1^0\|\|g_1) \oplus k_2^1$	$\rightarrow k_2^1$
	$H(k_0^1\|\|k_1^1\|\|g_1) \oplus k_2^1$	

Outline

1. Context
2. Security definitions
3. Oblivious transfer
4. Yao's original protocol
5. **Security improvements**
6. Performance improvements
7. Implementations
8. Conclusion

5. Security improvements

- Yao is only secure against *semi-honest* adversaries
- Areas for improvement
    1. Securing oblivious transfer
    2. Securing circuit construction
    3. Securing against corrupt inputs
- Remaining issues…

Securing oblivious transfer

- Problem with existing implementation:
  - Initially **P2** generates (k^{pub}, k^{prv}), ($k^{\perp}$, $\perp$)
  - **P1** can't verify that **P2** holds only one private key
  - **P2** can learn garbled values of 0 and 1 bits for **P2**'s input wires
- Allows for violations of *privacy* SFE principal in *malicious* case

Securing oblivious transfer

- Solution:
  - **P2** needs to provably bind itself from being able to decrypt both sent values
  - **P1** still cannot learn **P2**’s selected value

Securing oblivious transfer

P1 — $\mathbb{Z}_q^*$, generator g — P2

- Selects $C \in \mathbb{Z}_q^*$ such that **P2** does not know discrete log of C

$\xrightarrow{C}$

- Selects $i \in \{0, 1\}$
- Selects x_i, $0 \leq i < q-2$
- $\beta_i = g^{x_i}$, $\beta_{i-1} = C^*(g^{x_i})^{-1}$

$\xleftarrow{\beta_i,\ \beta_{i-1}}$

- Verifies that $\beta_i{}^*\beta_{i-1} = C$
- If so, proceed similarly to previous protocol

Securing circuit construction

- Problem with existing implementation:
  - **P1** can construct a garbled circuit that computes f' instead of f
  - f' could echo i_{p2} (or something more subtle)
  - **P1** could learn **P2**'s input
- Allows for violations of *privacy* SFE principal in *malicious* case

Securing circuit construction

- Zero-Knowledge Proofs
  - Too expensive for practical use
- Cut-and-Choose
  - **P1** garbles multiple circuits, **P2** checks some
  - Cat and mouse game

Cut-and-Choose v1.0

P1

P2

- Uniquely garbles *m* versions of the circuit

m circuits →

- Selects m-1 circuits to verify

← m-1 selections

- Un-garbles selected circuits

m-1 revealed circuits
i_{p1} for last circuit →

- Verifies m-1 circuits are correct

Protocol continues as normal

Cut-and-Choose v1.0

- Reduces **P1**'s chance to successfully cheat to $1/m$
- $1/m$ might not be enough security
- Verifying circuits is expensive, generating circuits is expensive
- Would be nice to get $\gg 1-(1/m)$ confidence for $\leq$ work

Cut-and-Choose v2.0

P1

P2

- Uniquely garbles *m* versions of the circuit

m circuits →

- Selects *m*/2 circuits to verify

← *m*/2 selections

- Un-garbles selected circuits

m/2 revealed circuits
m/2 garbled inputs →

- Verifies *m*/2 circuits
- Compute remaining *m*/2 circuits, abort if differences

Protocol continues as normal

Cut-and-Choose v2.0

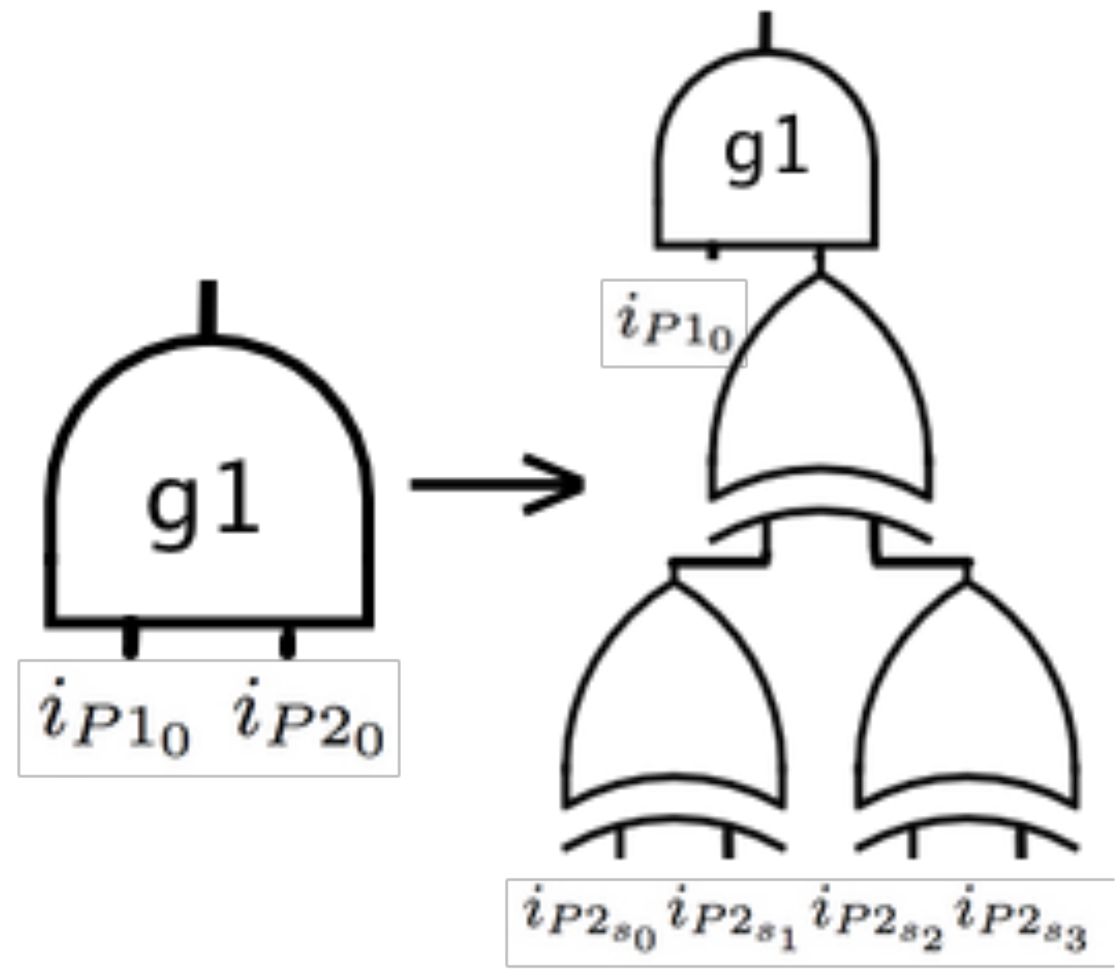
- **P1** will only succeed in attack if:
  - **P1** generates *m*/2 corrupt circuits
  - None of these *m*/2 circuits are among the *m*/2 **P2** selects to be revealed
- **P1**'s chance of success is tiny…
- But opens up a new early abort attack from **P1**…

Securing against corrupt inputs

- **P1** submits malicious input in OT:
  - 0 = valid garbled bit of i_{P2}, 1 = $\perp$
- If **P2** returns, $i_{P2_b} = 0$, if **P2** aborts, $i_{P2_b} = 1$
- P1 learns 1 bit of i_{P2}, violating *privacy* SFE principal

Securing against corrupt inputs

- Augment circuits with *s* additional input bits leading into XOR gates
- Gives **P2** 2^{s-1} ways to generate true desired input bit
- **P1** can still force abort, but learns nothing from it



Ensuring **P2** returns anything

- *Fairness* SFE principal requires that **P2** not be able to learn anything **P1** cannot

- No solutions to add this assurance to Yao

- Yao's protocol is not *fair*, and so not secure, in *malicious* case

- Focus on second best: ensuring that <u>if</u> **P2** does return, result is correct

    - Return encrypted values that **P1** has key for

    - Signature based solutions

Outline

1. Context
2. Security definitions
3. Oblivious transfer
4. Yao's original protocol
5. Security improvements
6. **Performance improvements**
7. Implementations
8. Conclusion

6. Performance improvements

- Yao’s protocol is “efficient” but expensive
- State of the art implementation takes <u>8 hours</u> to compute large string edit distance
- Billions of gates, gigs or more of memory per circuit

Areas for improvement

- Communication optimizations
- Execution optimizations
- Circuit optimizations

Communication optimizations

- Recall cut-and-check requires *m* circuits
- *m* circuits *
billions of gates *
4 multi byte values for each gate =
gigabytes to terabytes of overhead
- Can we do something about *m*?

Communication optimizations

- “Random Seed Checking”
- Don’t randomly assign keys
- Do so pseudo-randomly from initial random seed
- Instead of sending *m/2* verification circuits, **P1** send commitments of circuit construction and then initial random seed
- **P2** reconstructs circuit from random seed and checks that it matches the commitment

Execution optimizations

- Fast table lookups
- Pipelined circuit execution

Fast table lookups

Assume **P1**'s input is 1 and **P2**'s input is 0

	garbled value	
$H(k_0^{b_{p1}}\|\|k_0^{b_{p2}}\|\|g_1)\oplus$	$H(k_0^0\|\|k_1^0\|\|g_1)\oplus k_2^0$	$\rightarrow \perp$
$H(k_0^{b_{p1}}\|\|k_0^{b_{p2}}\|\|g_1)\oplus$	$H(k_0^0\|\|k_1^1\|\|g_1)\oplus k_2^1$	$\rightarrow \perp$
$H(k_0^{b_{p1}}\|\|k_0^{b_{p2}}\|\|g_1)\oplus$	$H(k_0^1\|\|k_1^0\|\|g_1)\oplus k_2^1$	$\rightarrow k_2^1$
	$H(k_0^1\|\|k_1^1\|\|g_1)\oplus k_2^1$	

---

w_0	w_1	w_2	garbled value
k_0^0	k_1^0	k_2^0	$H(k_0^0\|\|k_1^0\|\|g_1)\oplus(k_2^0\|\|index_b)$
k_0^0	k_1^1	k_2^1	$H(k_0^0\|\|k_1^1\|\|g_1)\oplus(k_2^1\|\|index_b)$
k_0^1	k_1^0	k_2^1	$H(k_0^1\|\|k_1^0\|\|g_1)\oplus(k_2^1\|\|index_b)$
k_0^1	k_1^1	k_2^1	$H(k_0^1\|\|k_1^1\|\|g_1)\oplus(k_2^1\|\|index_b)$

half index into next gate

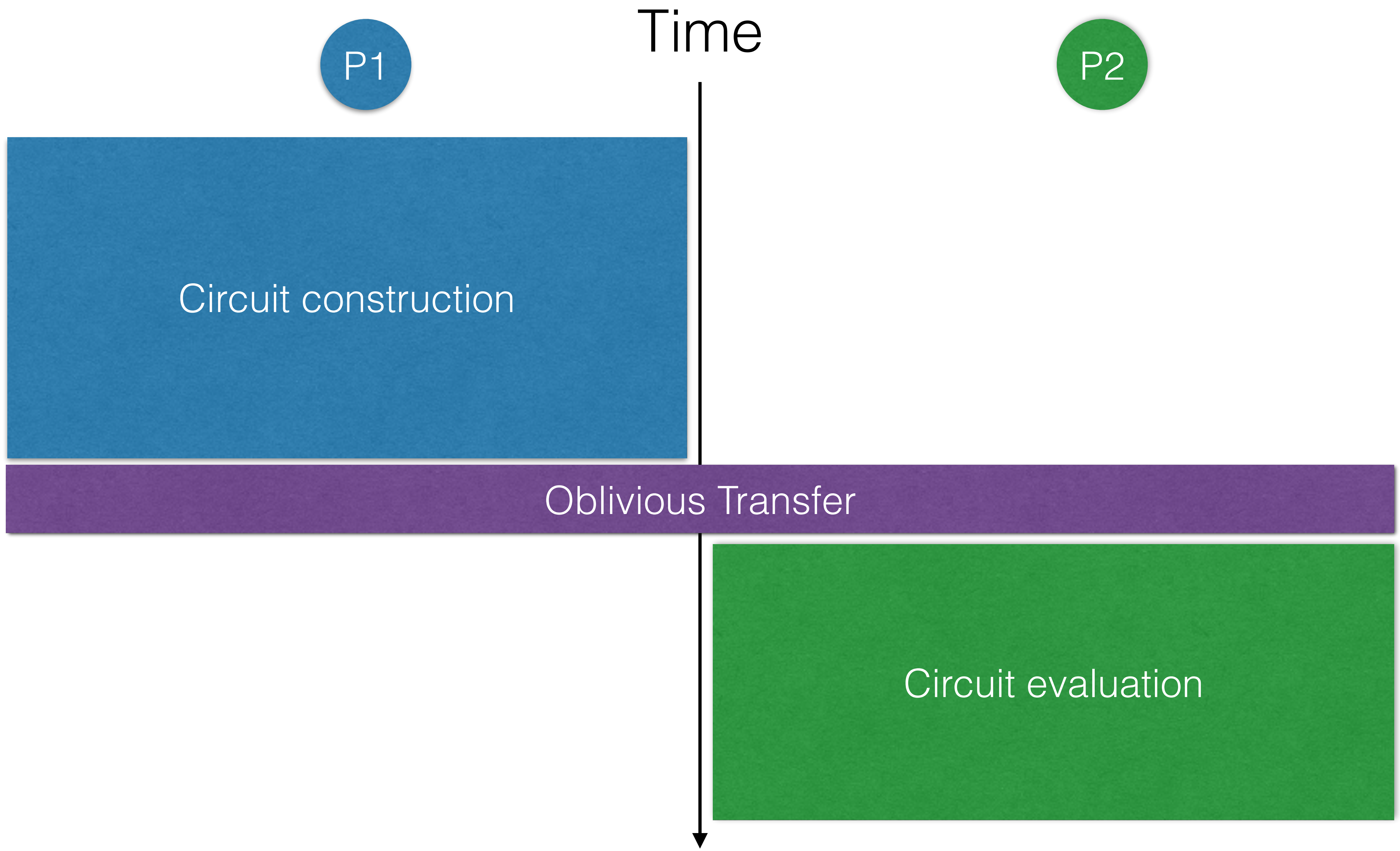
Fast table lookups

- Two index bits (one from each input wire) uniquely identify rows in each gate
- Slight increase in circuit construction cost
- Circuit execution now only needs one decryption per gate, instead of on average 2

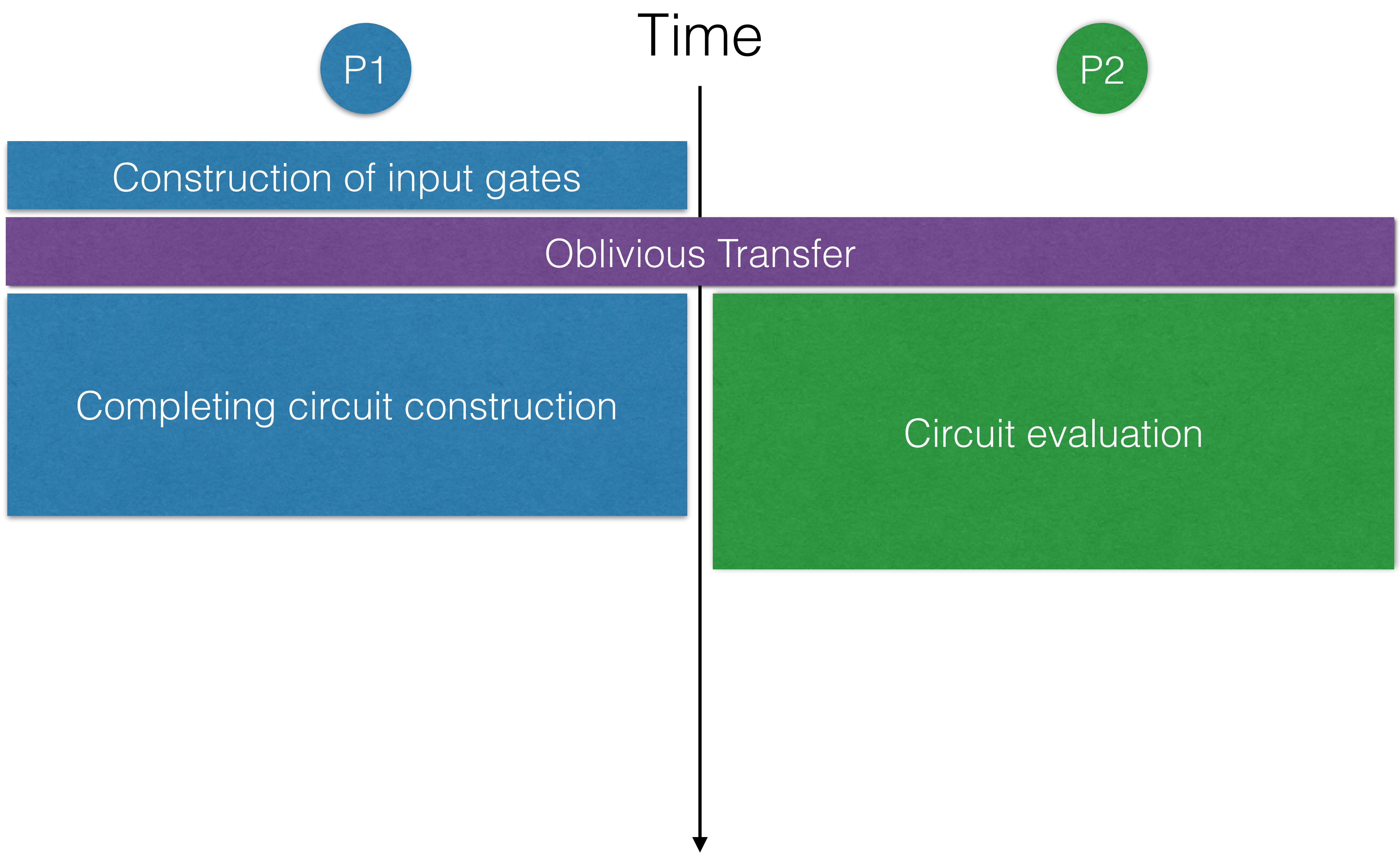
Pipelined circuit execution

- Standard version of Yao's protocol has
  - **P1** garbles, **P2** waits
  - **P2** evaluates, **P1** waits

Standard case



Pipelined circuit execution



Circuit optimizations

- Circuit simplification
- Free XORs
- "Garbled row reduction”

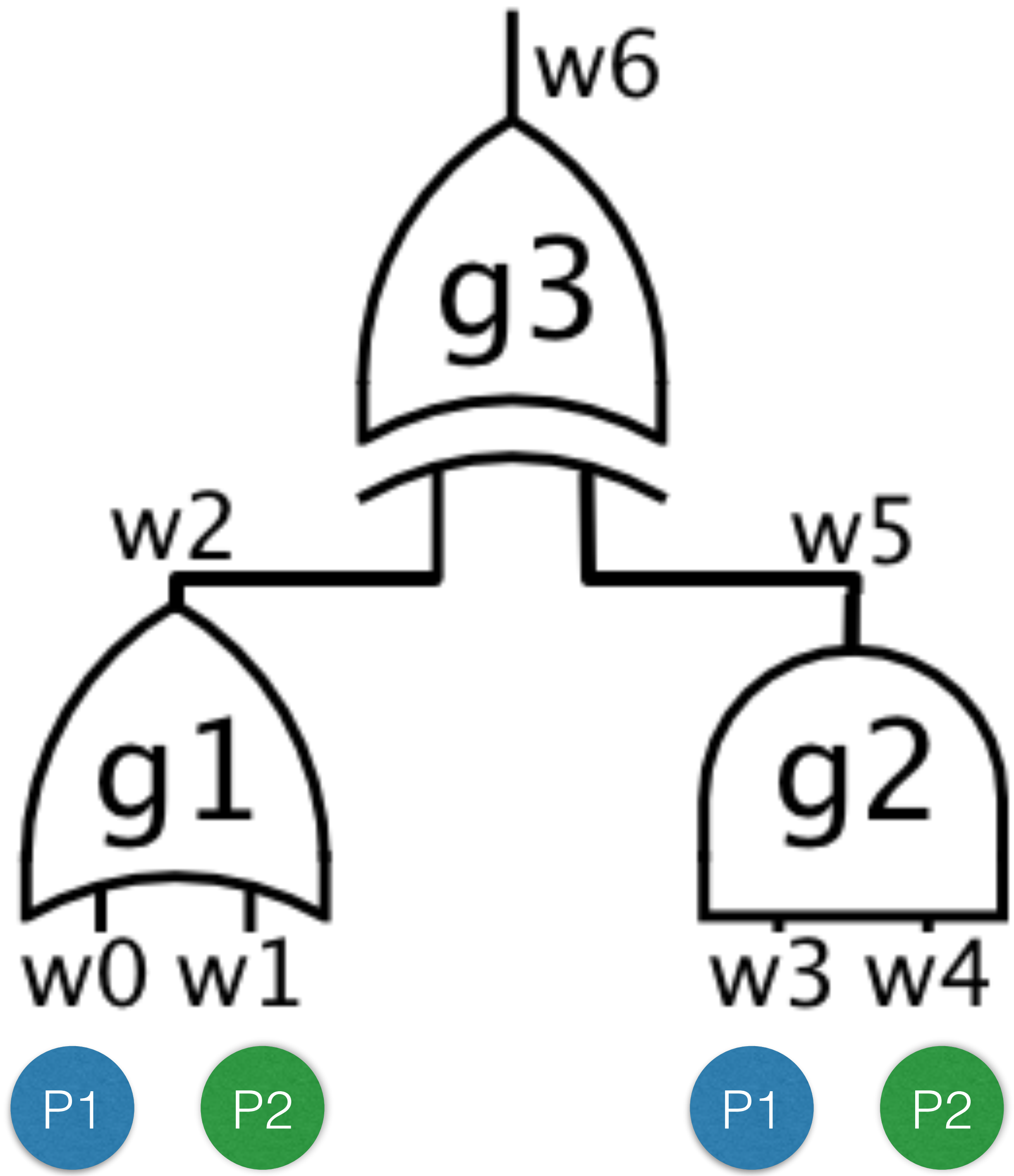
Circuit simplification

- removing errors in the f -> circuit conversion
- Remove dead chunks of the circuit
- Reduce sub-circuits that can be more efficiently represented by a smaller number of gates
- 60% reduction in circuit size for some circuit constructing tools (ex Fairplay)

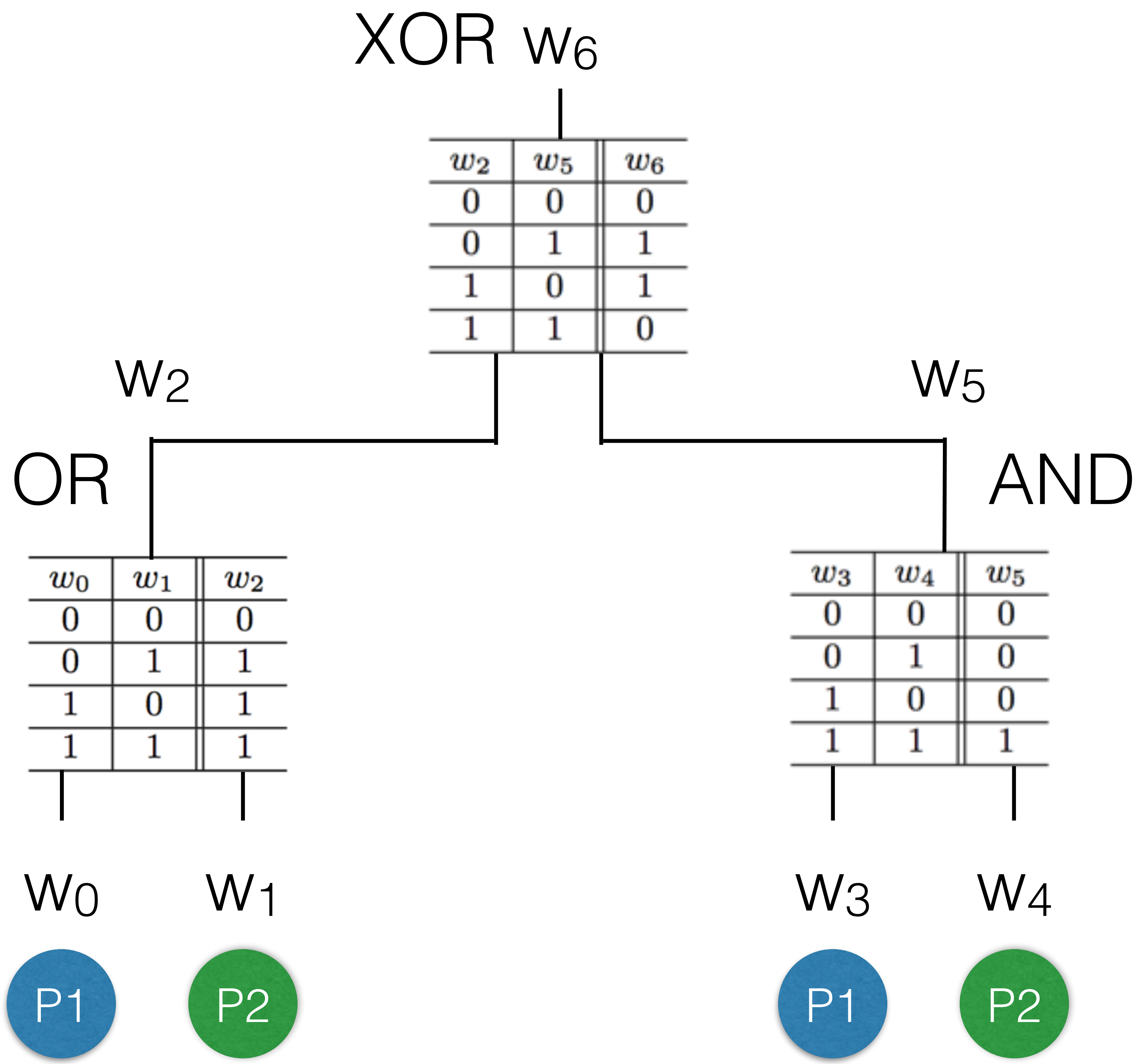
Free XORs

- By default all garbled values are independent
- Take advantage of this by fixing input values to XOR gates with single random R
- Replace XOR gates with an XOR function
- Remove 4 garbled values for each XOR gate

Free XORs



Free XORs



w_2	w_5	w_6
0	0	0
0	1	1
1	0	1
1	1	0

w_0	w_1	w_2
0	0	0
0	1	1
1	0	1
1	1	1

w_3	w_4	w_5
0	0	0
0	1	0
1	0	0
1	1	1

XOR

w_6

w_2	w_5	w_6	"unpacked"	"simplified"
k_2^0	k_5^0	$k_2^0 \oplus k_5^0$	$k_2^0 \oplus k_5^0$	$k_2^0 \oplus k_5^0$
k_2^0	k_5^1	$k_2^0 \oplus k_5^1$	$k_2^0 \oplus (k_5^0 \oplus R)$	$(k_2^0 \oplus k_5^0) \oplus R$
k_2^1	k_5^0	$k_2^1 \oplus k_5^0$	$(k_2^0 \oplus R) \oplus k_5^0$	$(k_2^0 \oplus k_5^0) \oplus R$
k_2^1	k_5^1	$k_2^1 \oplus k_5^1$	$(k_2^0 \oplus R) \oplus (k_5^0 \oplus R)$	$k_2^0 \oplus k_5^0$

w_2

w_5

OR

w_0	w_1	w_2
k_0^0	k_1^0	k_2^0
k_0^0	k_1^1	$k_2^1 = k_2^0 \oplus R$
k_0^1	k_1^0	$k_2^1 = k_2^0 \oplus R$
k_0^1	k_1^1	$k_2^1 = k_2^0 \oplus R$

AND

w_3	w_4	w_5
k_3^0	k_4^0	k_5^0
k_3^0	k_4^1	k_5^0
k_3^1	k_4^0	k_5^0
k_3^1	k_4^1	$k_5^1 = k_5^0 \oplus R$

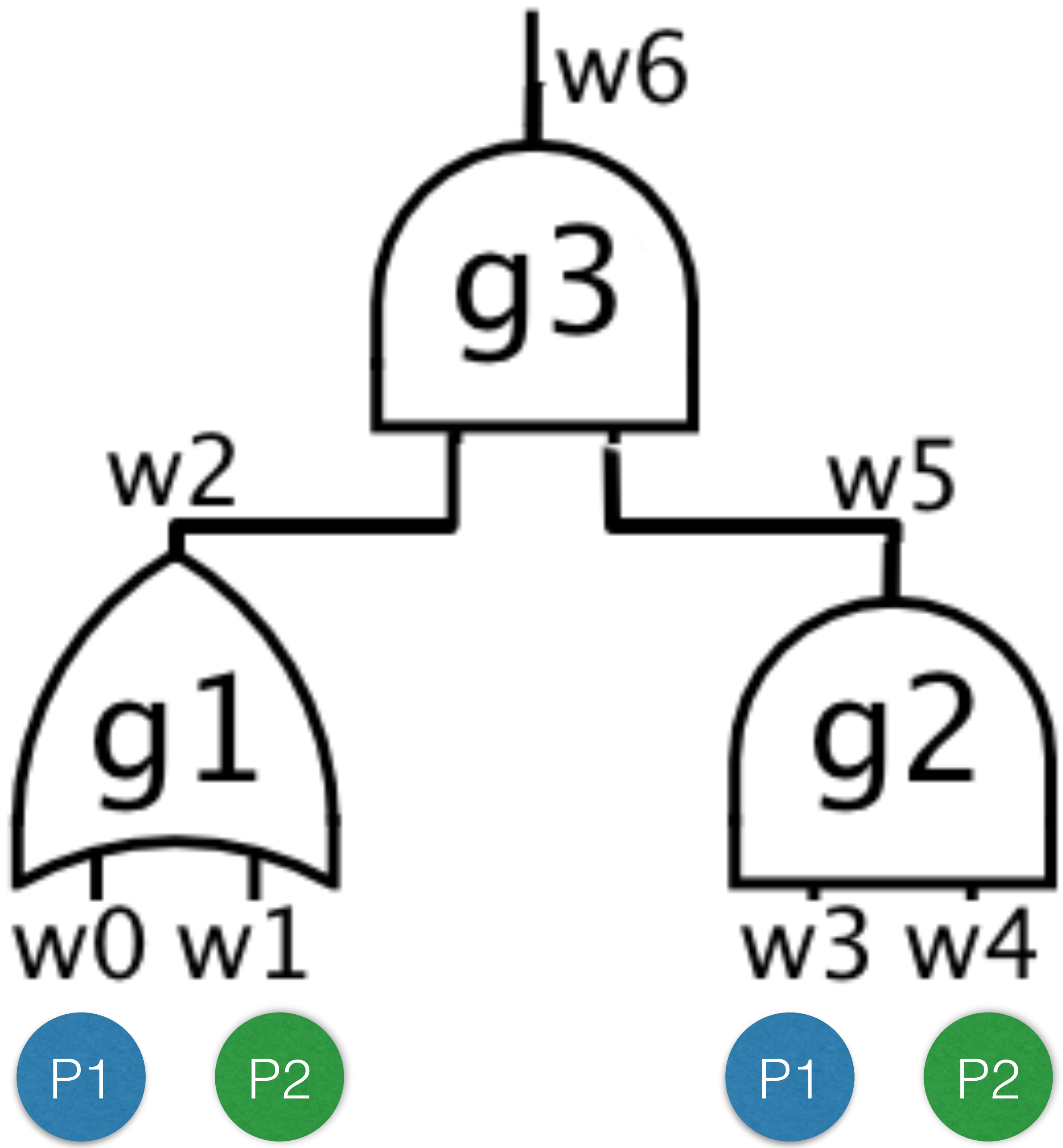
w_0 w_1 w_3 w_4

P1 P2 P1 P2

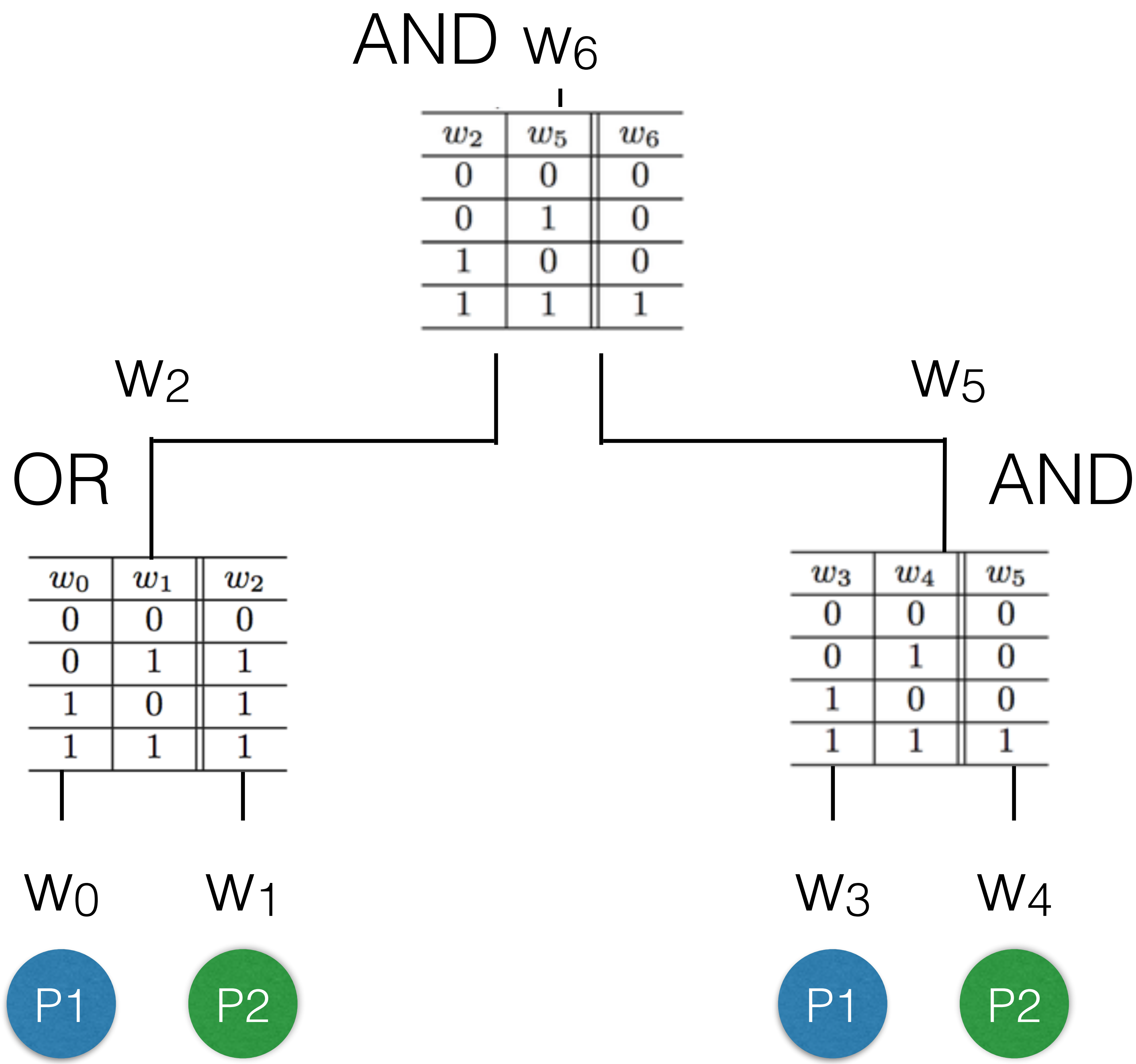
Garbled row reduction

- Similar to free XOR trick, but saves just one row
- Used for AND and OR gates
- Relies on the “fast table lookups” optimization
- Special cases garbled output value for one gate index, ex (0, 0)
- key is a function of input keys

Garbled row reduction



Garbled row reduction



w_2	w_5	w_6
0	0	0
0	1	0
1	0	0
1	1	1

w_0	w_1	w_2
0	0	0
0	1	1
1	0	1
1	1	1

w_3	w_4	w_5
0	0	0
0	1	0
1	0	0
1	1	1

Garbled row reduction

AND w_6

w_2	w_5	w_6	garbled value
k_2^0	k_5^0	k_6^0	$H(k_2^0 \| k_5^0 \| g_3) \oplus k_6^0$
k_2^0	k_5^1	k_6^0	$H(k_2^0 \| k_5^1 \| g_3) \oplus k_6^0$
k_2^1	k_5^0	k_6^0	$H(k_2^1 \| k_5^0 \| g_3) \oplus k_6^0$
k_2^1	k_5^1	k_6^1	$H(k_2^1 \| k_5^1 \| g_3) \oplus k_6^1$

w_2

w_5

OR

w_0	w_1	w_2	garbled value
k_0^0	k_1^0	k_2^0	$H(k_0^0 \| k_1^0 \| g_1) \oplus k_2^0$
k_0^0	k_1^1	k_2^1	$H(k_0^0 \| k_1^1 \| g_1) \oplus k_2^1$
k_0^1	k_1^0	k_2^1	$H(k_0^1 \| k_1^0 \| g_1) \oplus k_2^1$
k_0^1	k_1^1	k_2^1	$H(k_0^1 \| k_1^1 \| g_1) \oplus k_2^1$

AND

w_3	w_4	w_5	garbled value
k_3^0	k_4^0	k_5^0	$H(k_3^0 \| k_4^0 \| g_2) \oplus k_5^0$
k_3^0	k_4^1	k_5^0	$H(k_3^0 \| k_4^1 \| g_2) \oplus k_5^0$
k_3^1	k_4^0	k_5^0	$H(k_3^1 \| k_4^0 \| g_2) \oplus k_5^0$
k_3^1	k_4^1	k_5^1	$H(k_3^1 \| k_4^1 \| g_2) \oplus k_5^1$

w_0 w_1 w_3 w_4

P1 P2 P1 P2

Garbled row reduction

AND w6

w_2	w_5	w_6	garbled value
k_2^0	k_5^0	$H(k_2^0\|\|k_5^0\|\|g_3)$	$H(k_2^0\|\|k_5^0\|\|g_3)$
k_2^0	k_5^1	$H(k_2^0\|\|k_5^0\|\|g_3)$	$H(k_2^0\|\|k_5^1\|\|g_3) \oplus H(k_2^0\|\|k_5^0\|\|g_3)$
k_2^1	k_5^0	$H(k_2^0\|\|k_5^0\|\|g_3)$	$H(k_2^1\|\|k_5^0\|\|g_3) \oplus H(k_2^0\|\|k_5^0\|\|g_3)$
k_2^1	k_5^1	k_6^1	$H(k_2^1\|\|k_5^1\|\|g_3) \oplus k_6^1$

w2 w5

OR

w_0	w_1	w_2	garbled value
k_0^0	k_1^0	k_2^0	$H(k_0^0\|\|k_1^0\|\|g_1) \oplus k_2^0$
k_0^0	k_1^1	k_2^1	$H(k_0^0\|\|k_1^1\|\|g_1) \oplus k_2^1$
k_0^1	k_1^0	k_2^1	$H(k_0^1\|\|k_1^0\|\|g_1) \oplus k_2^1$
k_0^1	k_1^1	k_2^1	$H(k_0^1\|\|k_1^1\|\|g_1) \oplus k_2^1$

AND

w_3	w_4	w_5	garbled value
k_3^0	k_4^0	k_5^0	$H(k_3^0\|\|k_4^0\|\|g_2) \oplus k_5^0$
k_3^0	k_4^1	k_5^0	$H(k_3^0\|\|k_4^1\|\|g_2) \oplus k_5^0$
k_3^1	k_4^0	k_5^0	$H(k_3^1\|\|k_4^0\|\|g_2) \oplus k_5^0$
k_3^1	k_4^1	k_5^1	$H(k_3^1\|\|k_4^1\|\|g_2) \oplus k_5^1$

W0 W1 W3 W4

P1 P2 P1 P2

Outline

1. Context
2. Security definitions
3. Oblivious transfer
4. Yao’s original protocol
5. Security improvements
6. Performance improvements
7. **Implementations**
8. Conclusion

7. Implementations

- FairPlay (2004)
- Huang, Evans, Katz, Malka (2011)
- Kreuter, shelat, Shen (2012)

	Year	Security	Largest Circuit	Problems	Introduced Performance Optimizations
FairPlay	2004	Semi-Malicious	4.3k	Very simple	Fast Table Lookups Performance OT Protocols
Huang, et al.	2011	Semi-Honest	1 billion	Edit Distances AES	Free XORs Garbled Row Reduction Pipelined circuit execution
Kreuter, et al.	2012	Malcious	5.9 billion	AES RSA Signing Dot Product	Hardware optimizations Random seed checking Pipelining optimizations for above

Outline

1. Context
2. Security definitions
3. Oblivious transfer
4. Yao's original protocol
5. Security improvements
6. Performance improvements
7. Implementations
8. **Conclusion**

8. Conclusion

- Multi-party extensions for Yao
- Performance optimizing OT protocols
- Gateway to other areas
- much, much, much, much more…

Mission Accomplished

Any questions?