Collaborative Evaluation of Arbitrary Functions in a Constant Number of Rounds with Polynomial Amount of Communication

Secure Multiparty Computation

Philipp Müller

Technische Universität München

July 17, 2012

Part I

A short introduction to collaborative secure function evaluation

A motivational problem

- Several firms (numbered 1 to n) want to buy another firm
- Best bid wins!
- Participants only want to know *who* has best bid, *not how high* it was
- Mathematical formulation:

$$\underset{i \in \{1,2,\ldots,n\}}{\arg\max} \; \{x_i \mid x_i \text{ is bid of firm } i \in \{1, 2, \ldots, n\}\}$$

Problem

- Function accepting some arguments
- Each argument supplied by another party[1]
- Goal: Function evaluation, but keep arguments secret
- Possibly dishonest participants

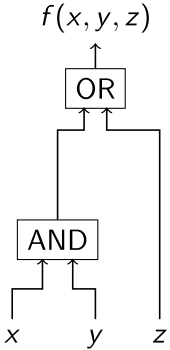
Example

$$f(x, y, z) = (x \wedge y) \vee z \tag{1}$$

[1] Also called player, participant.

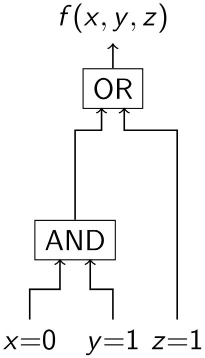
Functions as circuits

Idea: Represent functions as circuits



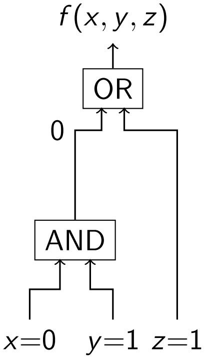
Functions as circuits

Idea: Represent functions as circuits



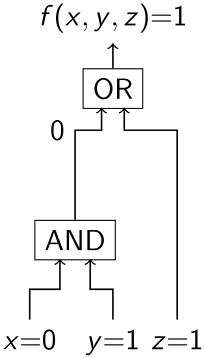
Functions as circuits

Idea: Represent functions as circuits



Functions as circuits

Idea: Represent functions as circuits



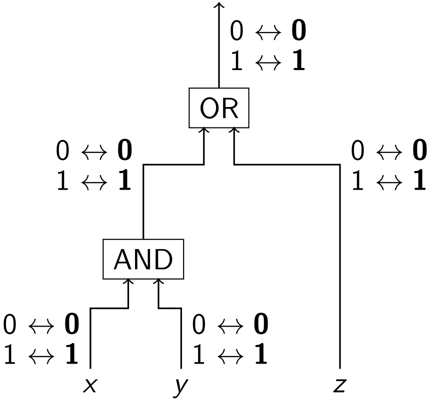
Problem

- "If we write a 0, it represents 0!"
- Each player can see/deduce everything
- Idea: "Ensure that each player sees some (random) stuff, but can't decide whether it's a 0 or a 1"
  $\Rightarrow$ Distinguish between *signals* and *plain-text*[2]
  $\Rightarrow$ Basic idea behind "garbled circuits"

[2]The plain-text is also called *semantics*, but I (try to) use the term plain-text since the term semantics is later used for something else.

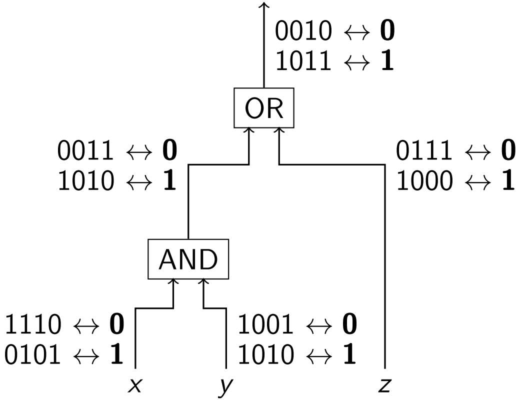
Garbled circuits – overview

- Up to now: Signal 0 means plain-text $\mathbf{0}$, signal 1 means $\mathbf{1}$
- Garbled circuit: Assign random signals to each wire!
- Idea: Hide plain-text by assigning random signals



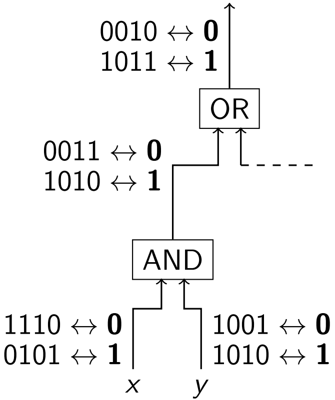
Garbled circuits – overview

- Up to now: Signal 0 means plain-text **0**, signal 1 means **1**
- Garbled circuit: Assign random signals to each wire!
- Idea: Hide plain-text by assigning random signals



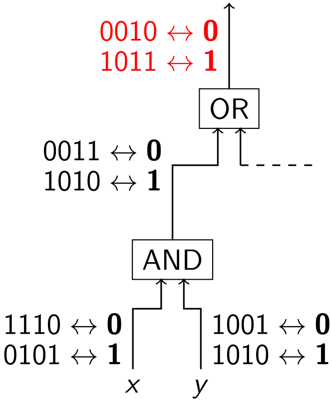
Garbled circuit – signals and plain-text

- *Odd* and *even* signals for each wire ("parity")
- One of those represents plain-text **0**, the other plain-text **1**
- Signals and mapping chosen randomly by all players
- Special case output wires: Even signal means **0**, odd means **1**



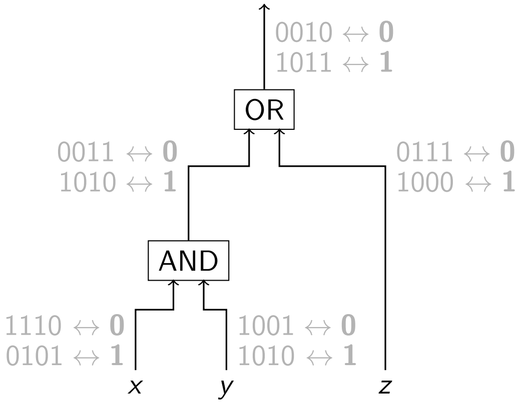
Garbled circuit – signals and plain-text

- *Odd* and *even* signals for each wire ("parity")
- One of those represents plain-text **0**, the other plain-text **1**
- Signals and mapping chosen randomly by all players
- Special case output wires: Even signal means **0**, odd means **1**



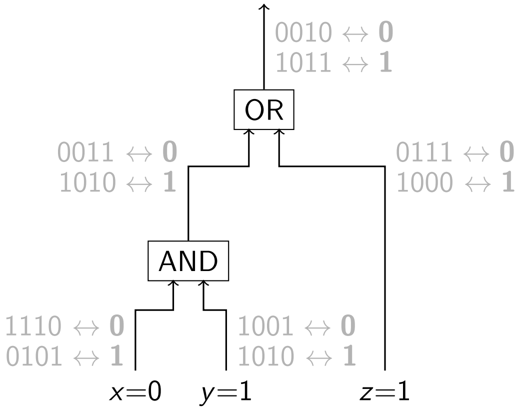
Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown
- This is everything a player should see



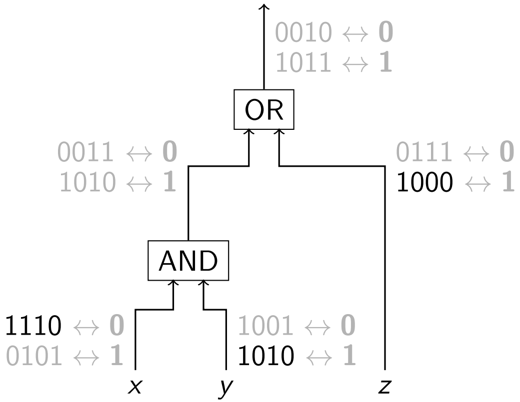
Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown
- This is everything a player should see



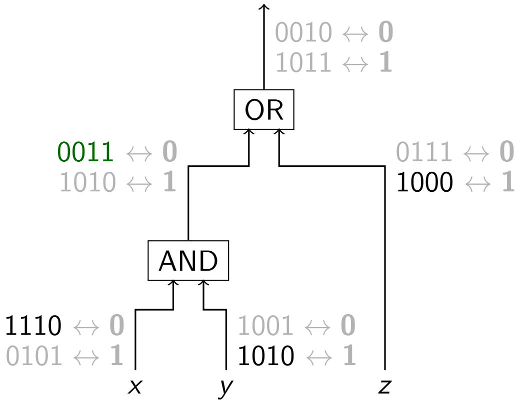
Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown
- This is everything a player should see



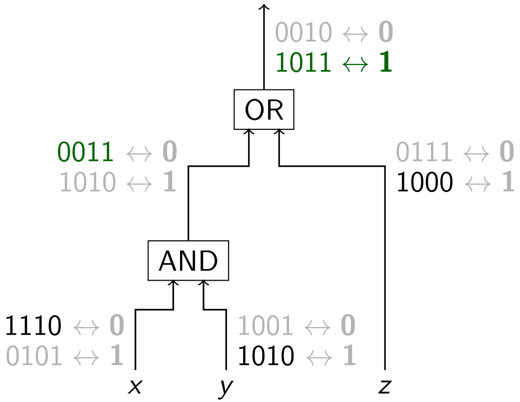
Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown
- This is everything a player should see



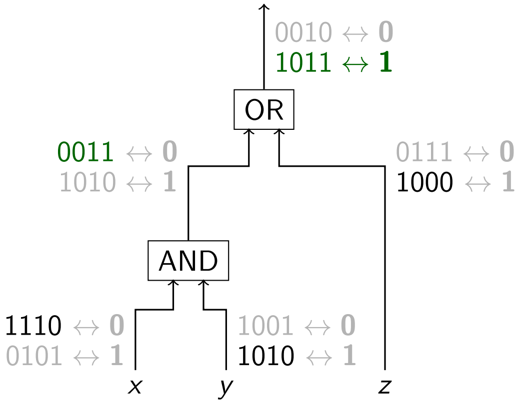
Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown
- This is everything a player should see



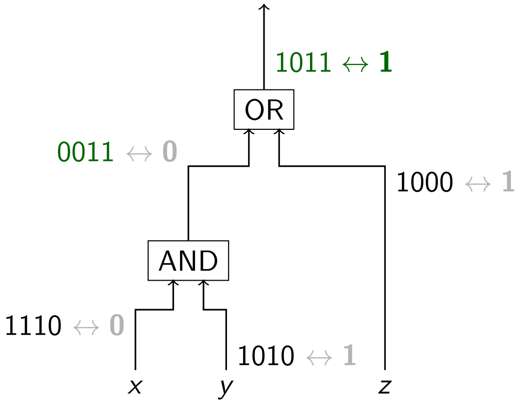
Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown
- This is everything a player should see



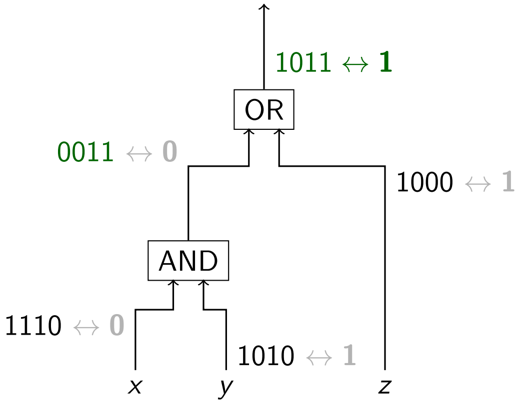
Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown
- This is everything a player should see



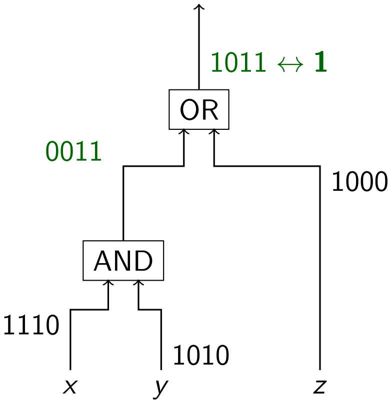
Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown, even the semantics
- This is everything a player should see



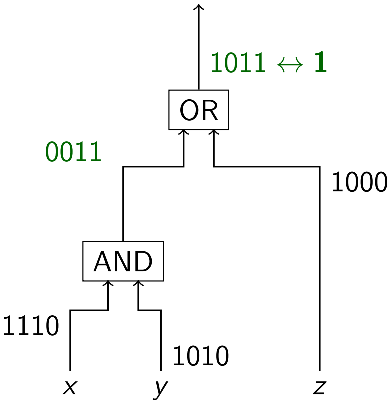
Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown, even the semantics
- This is everything a player should see

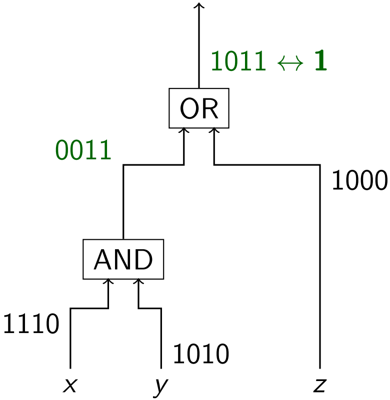


Our ultimate goal

- Compute *correct* output given only garbled circuit with inputs
- Everything else shall stay unknown, even the semantics
- This is everything a player should see



Our ultimate goal – outlook



- How to compute the signals?
- How to map from signals onto meanings?
- How to "implement" gates?

Model of computation – protocols

- Network of n players
- Private channels and broadcast channel
- Access to a fair coin
- Local computation "instant", communication expensive

Protocol:

- Certain "rules" for each player
- Organized in rounds
- May work in presence of malicious players (*adversaries*)
- Complexity measure: Rounds and communication
  $\Rightarrow$ Rounds are the limiting resource

Adversaries

- *One single* adversary. . .
- . . . that can infect several players
- Adversary can infect players at the beginning of each round as he wishes
- Adversary controls infected players fully
- Can infect less than half of the players

Our notion of security

Protocol shall:

- yield *correct* result in the presence of up to $\lfloor (n-1)/2 \rfloor$ dishonest participants
- compute the result securely (inputs shall not become public)
- not enable the adversary to deduce inputs.

Pseudorandom generators

- Deterministic, poly-time algorithm
- Takes a (truly random) string and stretches it to a longer string
- Output *indistinguishable* from truly random source
- Pseudorandom generators are one-way!

Pseudorandom generators G_0 and G_1

- Input: Binary string of length k
- Output: Binary string of length $\overline{n}k + 1$, where $\overline{n} = k^{10}$
- Number of players bounded by $\overline{n}$
- $\Rightarrow G_{\{0,1\}} : \{0,1\}^k \rightarrow \{0,1\}^{nk+1}$, with:
  - n: Number of players
  - k: Security parameter
- G_0 and G_1 are independent of each other

Pseudorandom generators G_0 and G_1

- Input: Binary string of length k
- Output: Binary string of length $\overline{n}k + 1$, where $\overline{n} = k^{10}$
- Number of players bounded by $\overline{n}$
- $\Rightarrow$ $G_{\{0,1\}} : \{0,1\}^k \to \{0,1\}^{nk+1}$, with:
  - n: Number of players
  - k: Security parameter
- G_0 and G_1 are independent of each other

Our basic building blocks

There are protocols to

- compute the XOR (denoted by the symbol $\oplus$) of an arbitrary number of bits
- evalutate any circuit of constant depth with bounded fan-in

in a constant number of rounds with polynomial[3] communication.
Proven secure for honest majority.

[3] W.r.t. circuit size and k

Our basic building blocks

There are protocols to

- compute the XOR (denoted by the symbol $\oplus$) of an arbitrary number of bits
- evalutate any circuit of constant depth with bounded fan-in

in a constant number of rounds with polynomial[3] communication.
Proven secure for honest majority.

[3] W.r.t. circuit size and k

Part II

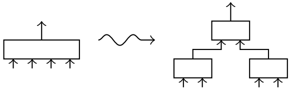
The protocol

Three phases in the protocol

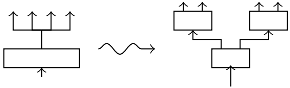
0. Preprocessing: Bring circuit in particular form
1. Players collaboratively compute garbled circuit and garbled input signals
2. Players locally evaluate garbled circuit on garbled input signals

Phase 0: suitable format of the circuit

0. No cycles, only gates of types AND, XOR, NEG, OR
1. Convert all gates to 2/1-gates ⇒ “binary tree construction”



2. Each wire is used as input wire *at most once* ⇒ introduce “splitters”



Remark

This increases the circuit’s size *only* by a polynomial factor.

Defining signals and semantics

- All players shall contribute to signals and semantics!
- Each player i generates the following:
  - For each wire ω, two random strings of length k: $s^{\omega}_{0,i}$ and $s^{\omega}_{1,i}$
  - For each wire ω except output wires, a random bit λ^{ω}_{i}

Remember:

Each wire shall obtain an even and an odd signal, that shall be randomly mapped onto $\{0, 1\}$ (mapping except for output wires).

Signals and semantics – definition

The k-bit string $s^{\omega}_{p,i}$ will be the contribution of player i to the *signal* of parity p for wire ω

The single bit λ^{ω}_{i} is the contribution of player i to the *semantics* of wire ω

More precisely:

Parity p signal for wire ω: $s^{\omega}_{p} := s^{\omega}_{p,1}\ s^{\omega}_{p,2}\ \ldots\ s^{\omega}_{p,n-1}\ s^{\omega}_{p,n}\ p$

$\Rightarrow$ Lenght of signals: $nk + 1$

Semantics of wire ω: $\lambda^{\omega} := \lambda^{\omega}_{1} \oplus \lambda^{\omega}_{2} \oplus \ldots \oplus \lambda^{\omega}_{n-1} \oplus \lambda^{\omega}_{n}$

Remark

These are just *definitions*.

Random signals and semantics – example

Three players and their random strings of length $k = 3$:

Player	$P1$	$P2$	$P3$
Contribution to s_0^ω	100	011	111
Contribution to s_1^ω	010	101	001
Contribution to λ^ω	0	0	1

Thus:

- $s_0^\omega = 100\,011\,111\,0$
- $s_1^\omega = 010\,101\,001\,1$
- $\lambda^\omega = 0 \oplus 0 \oplus 1 = 1$

$\lambda^\omega = 1$ means that $s_0^\omega \leftrightarrow \mathbf{1}$ and $s_1^\omega \leftrightarrow \mathbf{0}$

Things to keep in mind

- Each wire "gets" two signals s_0^ω and s_1^ω
  - Even and odd parity
  - One of them represents plaintext $\mathbf{0}$, the other one $\mathbf{1}$
- Signals and semantics are randomly constructed by all players.
- s_p^ω is the parity-p-signal for wire ω
- λ^ω is the *semantics* for wire ω
- Plain-text bit b on wire $\omega \Rightarrow$ choose signal with parity $b \oplus \lambda^\omega$, i.e. $s^\omega_{(b\oplus\lambda^\omega)}$

Things to keep in mind

- Each wire "gets" two signals s_0^ω and s_1^ω
  - Even and odd parity
  - One of them represents plaintext $\mathbf{0}$, the other one $\mathbf{1}$
- Signals and semantics are randomly constructed by all players.
- s_p^ω is the parity-p-signal for wire ω
- λ^ω is the *semantics* for wire ω
- Plain-text bit b on wire $\omega \Rightarrow$ choose signal with parity $b \oplus \lambda^\omega$, i.e. $s^\omega_{(b \oplus \lambda^\omega)}$

A word on semantics

- $\lambda^{\omega} = 0 \Leftrightarrow$ parity = plain-text
- $\lambda^{\omega} = 1 \Leftrightarrow$ parity = 1− plain-text

Example

$$\begin{array}{l} s_0^{\omega} = 100\,011\,111\,0 \\ s_1^{\omega} = 010\,101\,001\,1 \\ \\ \lambda^{\omega} = 1 \end{array}$$

ω

A word on semantics

- $\lambda^\omega = 0 \Leftrightarrow$ parity $=$ plain-text
- $\lambda^\omega = 1 \Leftrightarrow$ parity $= 1-$ plain-text

Example

$$s_0^\omega = 100\,011\,111\,0 \leftrightarrow \mathbf{1}$$
$$s_1^\omega = 010\,101\,001\,1 \leftrightarrow \mathbf{0}$$
$$\lambda^\omega = 1$$

ω

A word on semantics

- $\lambda^\omega = 0 \Leftrightarrow$ parity $=$ plain-text
- $\lambda^\omega = 1 \Leftrightarrow$ parity $= 1-$ plain-text

Example

$$s_0^\omega = 100\,011\,111\,0 \leftrightarrow \mathbf{1}$$
$$s_1^\omega = 010\,101\,001\,1 \leftrightarrow \mathbf{0}$$
$$\lambda^\omega = 1$$

$\uparrow \omega$

Plain-text $\mathbf{1}$ is represented by $s^\omega_{(\mathbf{1}\oplus\lambda^\omega)} = s^\omega_{\mathbf{1}\oplus 1} = s^\omega_0$

Computing the garbled inputs

- Input bit b: (Plain-text) bit along input wire ω
- For an input wire ω we set the signal

$$\sigma^{\omega} := s^{\omega}_{(b \oplus \lambda^{\omega})} \tag{2}$$

  - b: plain-text bit for wire ω
  - λ^{ω}: semantics for wire ω
  - $\Rightarrow b \oplus \lambda^{\omega}$: parity for the signal we need
  - $\Rightarrow s^{\omega}_{(b \oplus \lambda^{\omega})}$: proper garbled input signal

A word on garbled inputs

- Input signal $\sigma^\omega := s_p^\omega = s_{p,1}^\omega\ s_{p,2}^\omega\ \dots\ s_{p,n-1}^\omega\ s_{p,n}^\omega\ p$ with $p = b \oplus \lambda^\omega = b \oplus (\lambda_1^\omega \oplus \lambda_2^\omega \oplus \dots \oplus \lambda_{n-1}^\omega \oplus \lambda_n^\omega)$
- $b \oplus \lambda^\omega$: constant number of rounds, polynomial communication
- $\Rightarrow \sigma^\omega$: constant number of rounds, polynomial communication (it can be computed using a constant-depth circuit and some XORs)
- b is *not* revealed
- Signal for other parity stays secret

How can signals propagate along gates?

- We want to be able to evaluate circuit gate by gate
- $\Rightarrow$ Each gate has to choose the correct outgoing signal depending on its inputs
- $\Rightarrow$ provide some “help” to compute outgoing signals
  $\Rightarrow$ “gate labels”
- Incorporate input and output signals for that gate (input/output properly *defined*)
- “Ordinary” gates and splitter gates must be treated separately

Garbled signal propagation – ordinary gates

- Gate g computing a binary function $\otimes$ on bits
- Incoming wires α and β, outgoing wire γ
- What should be the output?
  - Parity of signal along α is a, parity of signal along β is b
  - $\Rightarrow$ Plain-text bits: $\lambda^\alpha \oplus a$, $\lambda^\beta \oplus b$
  - $\Rightarrow$ Plain-text result is $(\lambda^\alpha \oplus a) \otimes (\lambda^\beta \oplus b)$
- $\Rightarrow$ We need the garbled signal for wire γ that represents plaintext bit $(\lambda^\alpha \oplus a) \otimes (\lambda^\beta \oplus b)$
  $\Rightarrow$ This is exactly $s^\gamma_{[(\lambda^\alpha \oplus a) \otimes (\lambda^\beta \oplus b)] \oplus \lambda^\gamma}$

Garbled signal propagation – ordinary gates

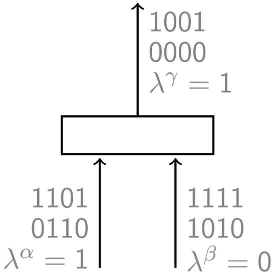
- Signals along input wires: $s_a^\alpha = s_{a,1}^\alpha \dots s_{a,n}^\alpha a$ and $s_b^\beta = s_{b,1}^\beta \dots s_{b,n}^\beta b$
- Signals along output wires: s_c^γ
- Split input signals into several subparts
- Apply a random generator onto the subparts

$$\begin{aligned} A_{ab}^g \quad &= \quad G_b(s_{a1}^\alpha) \oplus \cdots \oplus G_b(s_{an}^\alpha) \;\oplus \\ &\quad\;\; G_a(s_{b1}^\beta) \oplus \cdots \oplus G_a(s_{bn}^\beta) \;\oplus \\ &\quad\;\; s^\gamma_{[(\lambda^\alpha \oplus a)\otimes(\lambda^\beta \oplus b)]\oplus\lambda^\gamma} \\ &=: \quad G_b^*(s_a^\alpha) \oplus G_a^*(s_b^\beta) \oplus s^\gamma_{[(\lambda^\alpha \oplus a)\otimes(\lambda^\beta \oplus b)]\oplus\lambda^\gamma} \end{aligned} \tag{3}$$

- Compute *gate labels* for all parity combinations of a and b

Gate labels for ordinary gates – example

Parameters: $k = 1$, $n = 3$, $G_i(x) = xxxi$ (showcase PG)



$$\begin{aligned}
A^g_{10} &= G^*_0(s^\alpha_1) \oplus G^*_1(s^\beta_0) \oplus s^\gamma_{[(\lambda^\alpha \oplus 1)\wedge(\lambda^\beta \oplus 0)]\oplus\lambda^\gamma} \\
&= G^*_0(1101) \oplus G^*_1(1010) \oplus s^\gamma_{[(1\oplus 1)\wedge(0\oplus 0)]\oplus\lambda^\gamma} \\
&= G_0(1) \oplus G_0(1) \oplus G_0(0) \;\oplus \\
&\quad\; G_1(1) \oplus G_1(0) \oplus G_1(1) \;\oplus \\
&\quad\; s^\gamma_{[(1\oplus 1)\wedge(0\oplus 0)]\oplus 1} \\
&= 1110 \oplus 1110 \oplus 0000 \;\oplus \\
&\quad\; 1111 \oplus 0001 \oplus 1111 \;\oplus \\
&\quad\; s^\gamma_{[0\wedge 0]\oplus 1} \\
&= 0000 \oplus 0001 \oplus 1001 = 1000
\end{aligned}$$

Gate labels – ordinary gates

- Gate labels for gate g:

$$\begin{aligned} A^g_{ab} \quad = \quad & G_b(s^{\alpha}_{a1}) \oplus \cdots \oplus G_b(s^{\alpha}_{an}) \;\oplus \\ & G_a(s^{\beta}_{b1}) \oplus \cdots \oplus G_a(s^{\beta}_{bn}) \;\oplus \\ & s^{\gamma}_{[(\lambda^{\alpha}\oplus a)\otimes(\lambda^{\beta}\oplus b)]\oplus\lambda^{\gamma}} \end{aligned}$$

- Constant number of rounds, polynomial communication (constant-depth/bounded fan-in)
- Signals s^{α}_{ai} and s^{β}_{bj} cannot be deduced from the gate labels
- Neither can the semantics λ^{α}, λ^{β} and λ^{β}

Gate labels – splitter gates

- Completely analogue to ordinary gates
- Incoming wire α, outgoing wires γ_0, γ_1

$$A^g_{ab} \quad = \quad G_b(s^\alpha_{a1}) \oplus \cdots \oplus G_b(s^\alpha_{an}) \oplus s^{\gamma_b}_{(\lambda^\alpha \oplus a) \oplus \lambda^{\gamma_b}}$$

- Again, compute gate labels for all combinations of a and b
- Again computable in a constant number of rounds, polynomial communication

Phase 2: evaluating the garbled circuit

- Players computed:
  - Garbled input signals
  - Gate labels
- Players *individually* evaluate the garbled circuit gate by gate
- Compute gate output using gate labels

Evaluating an ordinary gate

- Given: Garbled gate g with signals σ^α and σ^β along input wires α and β
- Input signals carry parity a and b, respectively
- Reconsider equation (3):

$$A^g_{ab} = G^*_b(\sigma^\alpha_a) \oplus G^*_a(\sigma^\beta_b) \oplus \underbrace{\sigma^\gamma_{[(\lambda^\alpha \oplus a)\otimes(\lambda^\beta \oplus b)]\oplus\lambda^\gamma}}_{\text{Proper output}}$$

- We can solve to obtain the output:

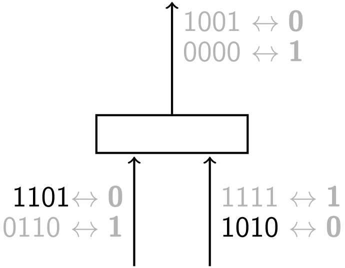
$$\sigma^\gamma = G^*_b(\sigma^\alpha) \oplus G^*_a(\sigma^\beta) \oplus A^g_{ab}$$

- Precise output computation:

$$\sigma^\gamma = G_b(\sigma^\alpha_1) \oplus \cdots \oplus G_b(\sigma^\alpha_n) \;\oplus\; G_a(\sigma^\beta_1) \oplus \cdots \oplus G_a(\sigma^\beta_n) \;\oplus\; A^g_{ab}$$

Evaluating an ordinary gate – example

Back to our previous example ($k = 1$, $n = 3$, $G_i(x) = xxxi$):

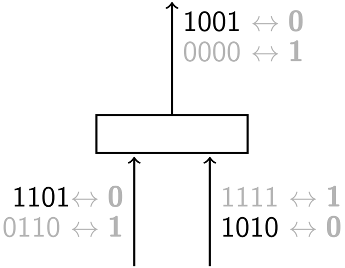


- Parities along α resp. β are $a = 1$ resp. $b = 0 \Rightarrow$ use gate label $A^g_{10} = 1000$ (computed before)
- Compute output:

$$\begin{aligned} \sigma^\gamma &= G^*_0(\sigma^\alpha_1) \oplus G^*_1(\sigma^\beta_0) \oplus A^g_{ab} \\ &= G^*_0(1101) \oplus G^*_1(1010) \oplus 1000 \\ &= 0000 \oplus 0001 \oplus 1000 = 1001 \end{aligned}$$

Evaluating an ordinary gate – example

Back to our previous example ($k = 1$, $n = 3$, $G_i(x) = xxxi$):



- Parities along α resp. β are $a = 1$ resp. $b = 0 \Rightarrow$ use gate label $A^g_{10} = 1000$ (computed before)
- Compute output:

$$\begin{aligned} \sigma^\gamma &= G_0^*(\sigma_1^\alpha) \oplus G_1^*(\sigma_0^\beta) \oplus A^g_{ab} \\ &= G_0^*(1101) \oplus G_1^*(1010) \oplus 1000 \\ &= 0000 \oplus 0001 \oplus 1000 = 1001 \end{aligned}$$

Evaluating a splitter gate

- Completely analogue technique
- $\Rightarrow$ Outputs along wires γ_0, γ_1 for a splitter gate:

$$\sigma^{\gamma_i} = G_i(\sigma_1^{\alpha}) \oplus \cdots \oplus G_i(\sigma_n^{\alpha}) \ \oplus A^{g}_{ai}$$

Outcome

- Collaborative construction
  - Constant number of rounds
  - Polynomial communication amount
  - Relies on existing sub-protocols
- Local garbled circuit evaluation without collaboration
- Use of pseudorandom generators and splitters hides plain-text

Strength of the protocoll:

- Random signals and semantics
- Even if we know the parity-0-signal, we cannot deduce the proper parity-1-signal
- Independence of gate labels, signals and semantics

Part III

Deferred or Omitted Explanations

Why I left it out

- A whole bunch of technical definitions
- No groundbreaking insights
- "Only" needed for formal verification of the protocol

Negligibility

Definition (Negligibility)

A function $\epsilon(k)$ is called *negligible*, if for all $c > 0$ there exists some k_0 such that $\epsilon(k) < k^{-c}$ for all $k > k_0$.

- In essence: Negligible if vanishing faster than any polynomial-inverse
- k is usually the *security parameter*
- In our case, k will (in some way) bound the number of participants
- Example: If the probability is negligible, we simply say: "That won't happen!"

Strings, ensembles

Definition

Given some alphabet Σ, we denote the set of all (finite-length) strings by Σ^*.

Definition (Ensemble)

A family of probability measures $\{\mathcal{A}_k\}$ on Σ^* is called an ensemble, if only strings of length at most $q(k)$ have positive probability of being picked, where $q(k)$ is some polynomial in k.

Indistinguishability

Definition (Indistinguishability)

For $\mathcal{A}$ taken from an ensemble and C a boolean circuit, $p_{\mathcal{A}}^{C}$ is the probability that C outputs 1 on an input randomly drawn according to distribution $\mathcal{A}$.

Ensembles $\mathcal{A}$ and $\mathcal{B}$ *computationally indistinguishable* if for any poly-size circuit family $\mathcal{C} = \{C_k\}$, the function $\epsilon(k) := |p_{\mathcal{A}_k}^{\mathcal{C}_k} - p_{\mathcal{B}_k}^{\mathcal{C}_k}|$ is negligible.

We call them *statistically indistinguishable* if

$$\epsilon(k) := \max_{S_k \subseteq \Sigma^*} |\Pr_{\mathcal{A}_k}[S_k] - \Pr_{\mathcal{B}_k}[S_k]|$$

is negligible, where $\Pr_{\mathcal{A}}[S]$ denotes the probability that we get a string within S if we draw randomly according to $\mathcal{A}$.

Notation: tagged vectors

- Vectors $\overrightarrow{v} = (v_1, \ldots, v_n)$ and $\overrightarrow{w} = (w_1, \ldots, w_n)$
- $T \subseteq \{1, 2, \ldots, n\}$
- $\overrightarrow{v}_T := \{(i, v_i) \mid i \in T\}$
- $\overline{T} = \{1, 2, \ldots, n\} \setminus T$
- $\overrightarrow{v}_T \cup \overrightarrow{w}_{\overline{T}}$ is the vector whose component indexed with indices from T are taken from $\overrightarrow{v}$, all other components are taken from $\overrightarrow{w}$

Oracles

Definition (t-bounded oracle)

A t-bounded $(\overrightarrow{x}, f)$-oracle is an oracle accepting two kinds of queries:

- Component query: A component query is an integer $i \in \{1, 2, \ldots, n\}$. If is only answered if t or fewer component queries were made so far. If this is the case, the oracle distinguishes:
  - If there was no output query so far, it is answered by x_i.
  - If there was already a proper output query (see below), namely $\overrightarrow{x}'_T$, the query is answered by $(x_i, f_i(\overrightarrow{x}_{\overline{T}} \cup \overrightarrow{x}'_T))$.
- Output query: An output query is a "tagged vector" $\overrightarrow{x}'_T$ (see below). It is answered by $f_T(\overrightarrow{x}_{\overline{T}} \cup \overrightarrow{x}'_T)$ if T consists precisely of the component queries made up to now and if there were not output queries so far. Further or improper output queries stay unanswered.

Some probability spaces

- Adverasry A's knowledge defines a probability space: $\mathbf{VIEW}_A^k(\overrightarrow{x})$
- Output of uncorrupted players: $\mathbf{OUTPUT}_A^k(\overrightarrow{x})$
- Output of a random algorithm S using an oracle: $\textsc{Output}\ S^{O_t(\overrightarrow{x},f)}(1^k)$
- $\textsc{Queries}\ S^{O_t(\overrightarrow{x},f)}(1^k)$ is a pair containing:
  - The indices i for which there was *never* a component query.
  - The (single) output query that was made by S.

Security

Definition (Privacy, Correctness)

Let $f : (\Sigma^l)^n \to (\Sigma^l)^n$. A protocol $\mathcal{P}$ t-securely computes f if for all t-adversaries A there exists a simulator S (probably using a t-bounded oracle) such that the following hold:

Privacy For all $\overrightarrow{x} \in (\Sigma^l)^n$, the k-parametrized ensemble $\mathbf{VIEW}_A^k(\overrightarrow{x})$ and the ensemble $\textsc{Output}\ S^{O_t(\overrightarrow{x},f)}(1^k)$ are computationally indistinguishable.

Correctness For all $\overrightarrow{x} \in (\Sigma^l)^n$, the k-parametrized ensembles $\mathbf{OUTPUT}_A^k(\overrightarrow{x})$ and $[(G, \overrightarrow{x}'_T) \leftarrow \textsc{Queries}\ S^{O_t(\overrightarrow{x},f)}(1^k) \mid f_G(\overrightarrow{x}_{\overline{T}} \cup \overrightarrow{x}'_T)]$ are statistically indistinguishable.

Proving the protocol secure

- We did not concretely specify a sub-protocol that computes gate labels and garbled input signals ...
- ... but we said that appropriate sub-protocols *exist*
- $\Rightarrow$ Quite a large amount of proof work is then done by the inventors of these sub-protocols
- Remaining proof constructs one simulator that works for *all* adversaries

Today no detailed proof:

- Original proof (without splitters, thus not completely correct) spans twenty pages and twelve sub-proofs
- Additional work for splitters (done about ten years later) involves another very non-trivial proof

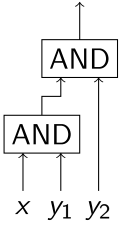
The need for pseudorandom generators

Why do we use them?

- Short answer: Because we need them!
- Long answer: No pseudorandom generators
  $\Rightarrow$ operate directly on (garbled) signals
  i.e. $A^g_{ab} = s^\alpha_a \oplus s^\beta_b \oplus s^\gamma_{[(a \oplus \lambda^\alpha) \otimes (b \oplus \lambda^\beta)] \oplus \lambda^\gamma}$
- $\Rightarrow$ We could deduce the plain-text values.

A simple attack

- Circuit $[(x \wedge y_1) \wedge y_2]$
- Assume I know that x is $\mathbf{0}$
- y_1, y_2 supplied another party
- $\Rightarrow$ Global output represents *surely* $\mathbf{0}$
- If there are no pseudorandom generators, we can deduce the plain-text of y_2



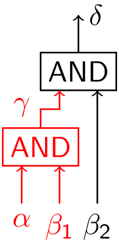
A simple attack – first AND-gate

Our knowledge up to now (wlog):

- $s_0^\alpha \leftrightarrow \mathbf{0}$ (we said x is 0)
- Signal along γ has parity 0, and represents plain-text $\mathbf{0}$
- σ_{ab}^γ: signal along wire γ for a parity-a-signal along α and a parity-b-signal along β_1

Now some calculations:

- $s_0^\gamma = A_{01} \oplus s_0^\alpha \oplus s_1^{\beta_1}$
- $\sigma_{10}^\gamma = A_{10} \oplus s_1^\alpha \oplus s_0^{\beta_1}$
- $\sigma_{11}^\gamma = A_{11} \oplus s_1^\alpha \oplus s_1^{\beta_1}$
- $\Rightarrow \sigma_{10}^\gamma \oplus \sigma_{11}^\gamma = A_{10} \oplus A_{11} \oplus s_0^{\beta_1} \oplus s_1^{\beta_1}$
- $\Rightarrow$ We know both signals and plain-text bits for γ



Known values:

$s_0^\alpha \leftrightarrow \mathbf{0}$

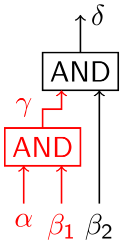
$s_0^{\beta_1}$

$s_0^\gamma \leftrightarrow \mathbf{0}$

A simple attack – first AND-gate

Our knowledge up to now (wlog):

- $s_0^\alpha \leftrightarrow \mathbf{0}$ (we said x is 0)
- Signal along γ has parity 0, and represents plain-text $\mathbf{0}$
- σ_{ab}^γ: signal along wire γ for a parity-a-signal along α and a parity-b-signal along β_1



Now some calculations:

- $s_0^\gamma = A_{01} \oplus s_0^\alpha \oplus s_1^{\beta_1}$
- $\sigma_{10}^\gamma = A_{10} \oplus s_1^\alpha \oplus s_0^{\beta_1}$
- $\sigma_{11}^\gamma = A_{11} \oplus s_1^\alpha \oplus s_1^{\beta_1}$
- $\Rightarrow \sigma_{10}^\gamma \oplus \sigma_{11}^\gamma = A_{10} \oplus A_{11} \oplus s_0^{\beta_1} \oplus s_1^{\beta_1}$
- $\Rightarrow$ We know both signals and plain-text bits for γ

Known values:

$s_0^\alpha \leftrightarrow \mathbf{0}$

$s_0^{\beta_1}$

$s_0^\gamma \leftrightarrow \mathbf{0}$

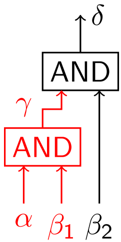
A simple attack – first AND-gate

Our knowledge up to now (wlog):

- $s_0^\alpha \leftrightarrow \mathbf{0}$ (we said x is 0)
- Signal along γ has parity 0, and represents plain-text $\mathbf{0}$
- σ_{ab}^γ: signal along wire γ for a parity-a-signal along α and a parity-b-signal along β_1

Now some calculations:

- $s_0^\gamma = A_{01} \oplus s_0^\alpha \oplus s_1^{\beta_1}$
- $\sigma_{10}^\gamma = A_{10} \oplus s_1^\alpha \oplus s_0^{\beta_1}$
- $\sigma_{11}^\gamma = A_{11} \oplus s_1^\alpha \oplus s_1^{\beta_1}$
- $\Rightarrow \sigma_{10}^\gamma \oplus \sigma_{11}^\gamma = A_{10} \oplus A_{11} \oplus s_0^{\beta_1} \oplus s_1^{\beta_1}$
- $\Rightarrow$ We know both signals and plain-text bits for γ



Known values:

$s_0^\alpha \leftrightarrow \mathbf{0}$

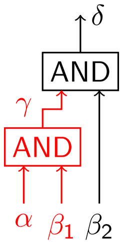
$s_0^{\beta_1}$ $s_1^{\beta_1}$

$s_0^\gamma \leftrightarrow \mathbf{0}$

A simple attack – first AND-gate

Our knowledge up to now (wlog):

- $s_0^\alpha \leftrightarrow \mathbf{0}$ (we said x is 0)
- Signal along γ has parity 0, and represents plain-text $\mathbf{0}$
- σ_{ab}^γ: signal along wire γ for a parity-a-signal along α and a parity-b-signal along β_1



Now some calculations:

- $s_0^\gamma = A_{01} \oplus s_0^\alpha \oplus s_1^{\beta_1}$
- $\sigma_{10}^\gamma = A_{10} \oplus s_1^\alpha \oplus s_0^{\beta_1}$
- $\sigma_{11}^\gamma = A_{11} \oplus s_1^\alpha \oplus s_1^{\beta_1}$
- $\Rightarrow \sigma_{10}^\gamma \oplus \sigma_{11}^\gamma = A_{10} \oplus A_{11} \oplus s_0^{\beta_1} \oplus s_1^{\beta_1}$
- $\Rightarrow$ We know both signals and plain-text bits for γ

Known values:

$s_0^\alpha \leftrightarrow \mathbf{0}$

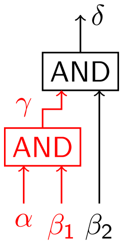
$s_0^{\beta_1}$ $s_1^{\beta_1}$

$s_0^\gamma \leftrightarrow \mathbf{0}$

A simple attack – first AND-gate

Our knowledge up to now (wlog):

- $s_0^\alpha \leftrightarrow \mathbf{0}$ (we said x is 0)
- Signal along γ has parity 0, and represents plain-text $\mathbf{0}$
- σ_{ab}^γ: signal along wire γ for a parity-a-signal along α and a parity-b-signal along β_1



Now some calculations:

- $s_0^\gamma = A_{01} \oplus s_0^\alpha \oplus s_1^{\beta_1}$
- $\sigma_{10}^\gamma = A_{10} \oplus s_1^\alpha \oplus s_0^{\beta_1}$
- $\sigma_{11}^\gamma = A_{11} \oplus s_1^\alpha \oplus s_1^{\beta_1}$
- $\Rightarrow \sigma_{10}^\gamma \oplus \sigma_{11}^\gamma = A_{10} \oplus A_{11} \oplus s_0^{\beta_1} \oplus s_1^{\beta_1}$
- $\Rightarrow$ We know both signals and plain-text bits for γ

Known values:

$s_0^\alpha \leftrightarrow \mathbf{0}$

$s_0^{\beta_1}$ $s_1^{\beta_1}$

$s_0^\gamma \leftrightarrow \mathbf{0}$ $s_1^\gamma \leftrightarrow \mathbf{1}$

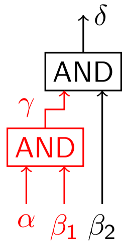
A simple attack – first AND-gate

Our knowledge up to now (wlog):

- $s_0^\alpha \leftrightarrow \mathbf{0}$ (we said x is 0)
- Signal along γ has parity 0, and represents plain-text $\mathbf{0}$
- σ_{ab}^γ: signal along wire γ for a parity-a-signal along α and a parity-b-signal along β_1

Now some calculations:

- $s_0^\gamma = A_{01} \oplus s_0^\alpha \oplus s_1^{\beta_1}$
- $\sigma_{10}^\gamma = A_{10} \oplus s_1^\alpha \oplus s_0^{\beta_1}$
- $\sigma_{11}^\gamma = A_{11} \oplus s_1^\alpha \oplus s_1^{\beta_1}$
- $\Rightarrow \sigma_{10}^\gamma \oplus \sigma_{11}^\gamma = A_{10} \oplus A_{11} \oplus s_0^{\beta_1} \oplus s_1^{\beta_1}$
- $\Rightarrow$ We know both signals and plain-text bits for γ



Known values:

$s_0^\alpha \leftrightarrow \mathbf{0}$

$s_0^{\beta_1}$ $s_1^{\beta_1}$

$s_0^\gamma \leftrightarrow \mathbf{0}$ $s_1^\gamma \leftrightarrow \mathbf{1}$

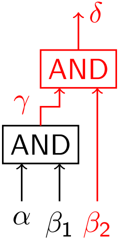
A simple attack – second AND-gate

We know by now (wlog):

- $s_0^\gamma \leftrightarrow \mathbf{0}, s_1^\gamma \leftrightarrow \mathbf{1}$ (signals and plain-text association)
- $s_0^{\beta_2}$ (input signal for plain-text bit of other player)
- Gate labels $B_{00}, B_{01}, B_{10}, B_{11}$

We compute $s_1^{\beta_2}$:

- $\sigma_{00}^\delta = B_{00} \oplus s_0^\gamma \oplus s_0^{\beta_2} \leftrightarrow \mathbf{0}$
- $\sigma_{01}^\delta = B_{01} \oplus s_0^\gamma \oplus s_1^{\beta_2} \leftrightarrow \mathbf{0}$
- $\Rightarrow$ Solve for $s_1^{\beta_2}$, and "test gate" with s_1^γ and $\{s_0^{\beta_2}, s_1^{\beta_2}\}$



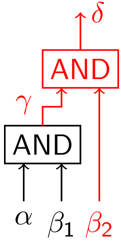
A simple attack – second AND-gate

We know by now (wlog):

- $s_0^\gamma \leftrightarrow \mathbf{0}, s_1^\gamma \leftrightarrow \mathbf{1}$ (signals and plain-text association)
- $s_0^{\beta_2}$ (input signal for plain-text bit of other player)
- Gate labels $B_{00}, B_{01}, B_{10}, B_{11}$

We compute $s_1^{\beta_2}$:

- $\sigma_{00}^\delta = B_{00} \oplus s_0^\gamma \oplus s_0^{\beta_2} \leftrightarrow \mathbf{0}$
- $\sigma_{01}^\delta = B_{01} \oplus s_0^\gamma \oplus s_1^{\beta_2} \leftrightarrow \mathbf{0}$
- $\Rightarrow$ Solve for $s_1^{\beta_2}$, and "test gate" with s_1^γ and $\{s_0^{\beta_2}, s_1^{\beta_2}\}$



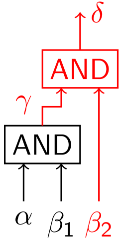
A simple attack – second AND-gate

We know by now (wlog):

- $s_0^\gamma \leftrightarrow \mathbf{0}, s_1^\gamma \leftrightarrow \mathbf{1}$ (signals and plain-text association)
- $s_0^{\beta_2}$ (input signal for plain-text bit of other player)
- Gate labels $B_{00}, B_{01}, B_{10}, B_{11}$

We compute $s_1^{\beta_2}$:

- $\sigma_{00}^\delta = B_{00} \oplus s_0^\gamma \oplus s_0^{\beta_2} \leftrightarrow \mathbf{0}$
- $\sigma_{01}^\delta = B_{01} \oplus s_0^\gamma \oplus s_1^{\beta_2} \leftrightarrow \mathbf{0}$
- $\Rightarrow$ Solve for $s_1^{\beta_2}$, and "test gate" with s_1^γ and $\{s_0^{\beta_2}, s_1^{\beta_2}\}$



How do pseudorandom generators help?

Without pseudorandom generators:

- Plain-text values computable

With pseudorandom generators:

- We can deduce the values $G_a^*(\sigma^\beta)$ ...
- ... but *not* the values σ^β
- $\Rightarrow$ We do not know the proper other signal for wire β
- $\Rightarrow$ We cannot try all signal combinations

Essence:

Applying a pseudorandom generator onto the signals prevents us from deducing the proper signal for the complementary parity[4]!

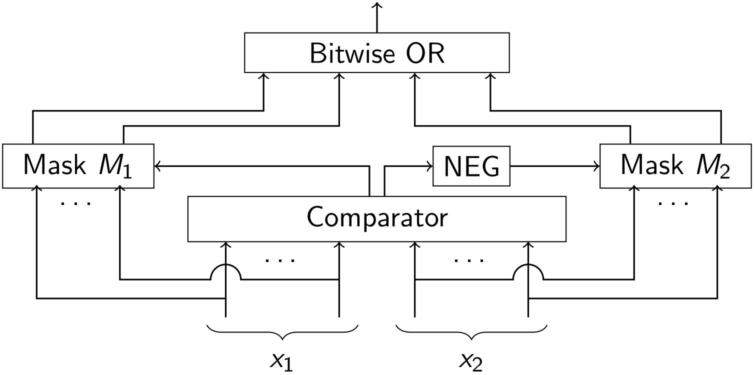
[4] And thus, from deducing plain-text values.

Why do we need splitters?

- Without splitters, different gates are dependent
- $\Rightarrow$ This enables us to determine plain-text bits with high probability

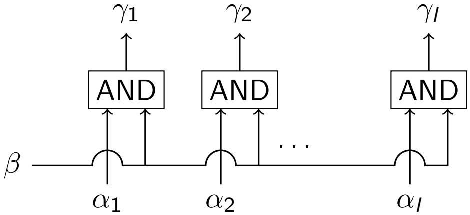
An attack on splitter-free circuits

A comparator for two l-bit numbers:



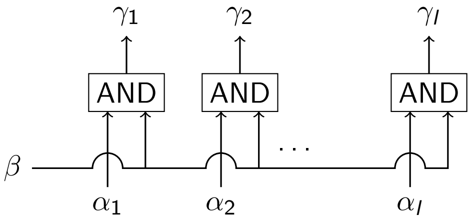
- We assume that an earlier execution yielded $x_2 \geq x_1$
- How can player 2 exploit this?
- $\Rightarrow$ Important part: Mask M_1

An attack on splitter-free circuits



- What would happen if x_1 was 0?
- Assume signals along α_i represents $\mathbf{0}$
- Assume that signal along β represents $\mathbf{1}$
- We are able to deduce the correct values of x_1 with high probability
- This can be acchieved a probabilistic algorithm

An attack on splitter-free circuits



- Assume that $\sigma^{\alpha_i} \leftrightarrow \mathbf{0}$ for all $i \in \{1, 2, \ldots, l\}$ (likely wrong)
- We know that $\sigma^{\gamma_i} \leftrightarrow \mathbf{0}$
- Idea: For all correctly-guessed bits, we get "useful results", for the wrongly-guessed bits, we obtian random stuff.
- Gate i:

$$\sigma^{\gamma_i} = A^i_{ab} \oplus \underbrace{G^*_b(\sigma^{\alpha_i}_{a_i})}_{G_b(\sigma^{\alpha_i}_{a_i,1}) \oplus G_b(\sigma^{\alpha_i}_{a_i,2})} \oplus \underbrace{G^*_{a_i}(\sigma^{\beta}_{b})}_{G_{a_i}(\sigma^{\beta}_{b,1}) \oplus G_{a_i}(\sigma^{\beta}_{b,2})}$$

- $G^*_b(\sigma^{\alpha_i}_{a_i})$ present in *all* gates

An attack on splitter-free circuits

- Gate i:

$$\sigma^{\gamma_i} = A^i_{ab} \oplus \underbrace{G^*_b(\sigma^{\alpha_i}_{a_i})}_{G_b(\sigma^{\alpha_i}_{a_i,1}) \oplus G_b(\sigma^{\alpha_i}_{a_i,2})} \oplus \underbrace{G^*_{a_i}(\sigma^{\beta}_{b})}_{G_{a_i}(\sigma^{\beta}_{b,1}) \oplus G_{a_i}(\sigma^{\beta}_{b,2})}$$

- $G^*_{a_i}(\sigma^{\beta}_{b})$ present in *all* gates
- "Solve" for $G^*_{a_i}(\sigma^{\beta}_{b})$ and rename result:

$$\mu_i := \sigma^{\gamma_i} \oplus G_b(s^{\alpha_i}_{a,1}) \oplus G_b(s^{\alpha_i}_{a,2}) \oplus A^{g_i}_{a_i b}$$

- If our guess for bit i was correct, μ_i corresponds to $G^*_{a_i}(\sigma^{\beta}_{b})$, otherwise it is a random string

An attack on splitter-free circuits

- Collection of μ_i's
- Correct guesses:
  $\Rightarrow$ either $G_0(s^\beta_{b1}) \oplus G_0(s^\beta_{b2})$ or $G_1(s^\beta_{b1}) \oplus G_1(s^\beta_{b2})$
- Incorrect guesses: Random strings
- $\Rightarrow$ group the μ_i's (random vs. $G_0(s^\beta_{b1}) \oplus G_0(s^\beta_{b2})$ or $G_1(s^\beta_{b1}) \oplus G_1(s^\beta_{b2})$)
- $\Rightarrow$ random strings correspond to wrongly-guessed bits $\Rightarrow$ these bits are probably **1**, all the others are probably **0**

Part IV

Literature

For further reading: original protocol description

Rogaway, P.:
The Round Complexity Of Secure Protocols.
Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA (1991)

Beaver, D., Micali, S., Rogaway, P.:
The round complexity of secure protocols.
In: Proceedings of the twenty-second annual ACM symposium on Theory of computing, New York, NY, USA, ACM (1990) 503–513

For further reading: splitter construction

Tate, S.R., Xu, K.:
On Garbled Circuits and Constant Round Secure Function Evaluation.
Technical report, UNIVERSITY OF NORTH TEXAS (2003)

Xu, K.:
Mobile agent security through multi-agent cryptographic protocols.
PhD thesis, University of North Texas, Denton, TX, USA (2004) AAI3126591.

For further reading: a modern treatment of garbled circuits

Bellare, M., Hoang, V.T., Rogaway, P.:
Garbling schemes.
Cryptology ePrint Archive, Report 2012/265 (2012)
http://eprint.iacr.org/.

For further reading: general crypto readings

Goldwasser, S., Bellare, M.:
Lecture notes on cryptography.
Lecture Notes (July 2008) Accessed March 2012
(http://cseweb.ucsd.edu/users/mihir/papers/gb.pdf).

Goldreich, O.:
Foundations of Cryptography: Volume 2, Basic Applications.
Cambridge University Press, New York, NY, USA (2004)

For further reading: pseudorandom generators and one-way functions

Blum, M., Micali, S.:
How to generate cryptographically strong sequences of pseudo-random bits.
SIAM J. Comput. **13**(4) (November 1984) 850–864

Yao, A.C.:
Theory and application of trapdoor functions.
In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science. SFCS '82, Washington, DC, USA, IEEE Computer Society (1982) 80–91

For further reading: other readings I

Shamir, A.:
How to share a secret.
Commun. ACM **22**(11) (November 1979) 612–613

Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.:
Verifiable secret sharing and achieving simultaneity in the presence of faults.
In: Proceedings of the 26th Annual Symposium on Foundations of Computer Science. SFCS '85, Washington, DC, USA, IEEE Computer Society (1985) 383–395

For further reading: other readings II

Rabin, T., Ben-Or, M.:
Verifiable secret sharing and multiparty protocols with honest majority.
In: Proceedings of the twenty-first annual ACM symposium on Theory of computing. STOC '89, New York, NY, USA, ACM (1989) 73–85

Beaver, D.:
Multiparty protocols tolerating half faulty processors.
In: Proceedings on Advances in cryptology. CRYPTO '89, New York, NY, USA, Springer-Verlag New York, Inc. (1989) 560–572

For further reading: other readings III

- Goldreich, O., Micali, S., Wigderson, A.:
  How to play any mental game.
  In: Proceedings of the nineteenth annual ACM symposium on Theory of computing. STOC '87, New York, NY, USA, ACM (1987) 218–229

- Ben-Or, M., Goldwasser, S., Wigderson, A.:
  Completeness theorems for non-cryptographic fault-tolerant distributed computation.
  In: Proceedings of the twentieth annual ACM symposium on Theory of computing. STOC '88, New York, NY, USA, ACM (1988) 1–10