Advanced Cryptography
CS 655

Week 12:

- Functional Secret Sharing/Distributed Point Functions

Save the Date: Midwest CRYPTO Day on April 11^{th} at UIUC

Spring 2023

Secret Sharing

- (t,n)-Secret Sharing
  - $[\![s]\!]_1, [\![s]\!]_2, \ldots, [\![s]\!]_n = \text{ ShareGen}(\mathrm{s}, \mathrm{t}, \mathrm{n})$
    - Takes as input a secret an outputs n shares
  - $\mathrm{s} = \text{RecoverShares}\left(\left(i_1, [\![s]\!]_{i_1}\right), \left(i_2, [\![s]\!]_{i_2}\right), \ldots, \left(i_t, [\![s]\!]_{i_t}\right)\right)$
    - Takes as input a subset of t distinct shares and outputs the secret s

- **Information Theoretic Privacy:** any subset of t-1 shares leaks no information about the secret s

$$\Pr\left[[\![s]\!]_{i_1}, \ldots, [\![s]\!]_{i_{t-1}} | s\right] = \Pr\left[[\![s]\!]_{i_1}, \ldots, [\![s]\!]_{i_{t-1}} | s'\right]$$

Secret Sharing

- (t,n)-Secret Sharing
  - $[\![s]\!]_1, [\![s]\!]_2, \dots, [\![s]\!]_n = \text{ ShareGen}(\mathrm{s}, \mathrm{t}, \mathrm{n})$
    - Takes as input a secret an outputs n shares
  - $\mathrm{s} = \text{Recover}\left(\left(i_1, [\![s]\!]_{i_1}\right), \left(i_2, [\![s]\!]_{i_2}\right), \dots, \left(i_t, [\![s]\!]_{i_t}\right)\right)$
    - Takes as input a subset of t distinct shares and outputs the secret s

- **Example 1:** (n,n)-Secret Sharing for secrets $s \in \{0,1\}^\lambda$
  - $\text{ShareGen}(\mathrm{s}, \mathrm{t}, \mathrm{n})$
    - Pick $[\![s]\!]_1, [\![s]\!]_2, \dots, [\![s]\!]_{n-1} \in \{0,1\}^\lambda$ uniformly at random
    - Compute $[\![s]\!]_n = s \oplus [\![s]\!]_1 \oplus [\![s]\!]_2 \oplus \dots \oplus [\![s]\!]_{n-1}$
  - $\text{Recover}([\![s]\!]_1, [\![s]\!]_2, \dots, [\![s]\!]_n) = [\![s]\!]_1 \oplus [\![s]\!]_2 \oplus \dots \oplus [\![s]\!]_n$

Shamir Secret Sharing

- Uses polynomials over a field $\mathbb{F}$
- **Fact:** Suppose that $\mathrm{p(x)} = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ is a polynomial over a field $|\mathbb{F}| > t$ and let $x_1, \dots, x_t$ be any set of t distinct points on the field. Then the polynomial $\mathrm{p(x)}$ is uniquely determined by the outputs $\left(\mathrm{p}(x_1), \dots, p(x_t)\right)$.

Proof Sketch: If there is another degree $t-1$ polynomial

$$f(x) = b_0 + b_1 x + \cdots + b_{t-1} x^{t-1}$$

such that $\left(\mathrm{p}(x_1), \dots, p(x_t)\right) = \left(\mathrm{f}(x_1), \dots, f(x_t)\right)$ then the polynomial

$$g(x) \coloneqq f(x) - p(x) = (b_0 - a_0) + (b_1 - a_1)x + \cdots + (b_{t-1} - a_{t-1})x^{t-1}$$

has t roots i.e., $\mathrm{g}(x_i) = f(x_i) - p(x_i) = 0$. But $g(x)$ has degree at most $t-1$ which means that it can have at most $t-1$ roots. Contradiction!

Shamir Secret Sharing

- Uses polynomials over a field $\mathbb{F}$
- **Fact:** Suppose that $\mathrm{p(x)} = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ is a polynomial over a field $|\mathbb{F}| > t$ and let $x_1, \dots, x_t$ be any set of t distinct points on the field. Then the polynomial $\mathrm{p(x)}$ is uniquely determined by the outputs $\left(\mathrm{p}(x_1), \dots, p(x_t)\right)$.

Lagrange Interpolation: Efficient algorithm to find coefficients of $\mathrm{p(x)}$ given $x_1, \dots, x_t$ and $\left(\mathrm{p}(x_1), \dots, p(x_t)\right)$

Shamir Secret Sharing

- Uses polynomials over a field $\mathbb{F}$
- **Fact:** Suppose that $\mathrm{p(x)} = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ is a polynomial over a field $|\mathbb{F}| > t$ and let $x_1, \ldots, x_t$ be any set of t distinct points on the field. Then the polynomial $\mathrm{p(x)}$ is uniquely determined by the outputs $\left(\mathrm{p}(x_1), \ldots, p(x_t)\right)$

- View secret $s \in \mathbb{F}$ as a field element
  - Fix distinct field elements $x_0, \ldots, x_{n-1} \in \mathbb{F}$
  - $\mathrm{ShareGen(s, t, n)}$
    - Pick $[\![s]\!]_1, [\![s]\!]_2, \ldots, [\![s]\!]_{t-1} \in \mathbb{F}$ uniformly at random
    - Define the polynomial $f(x)$ such that $\left(\mathrm{f}(x_0), \ldots, f(x_{t-1})\right) = (\mathrm{s}, [\![s]\!]_1, [\![s]\!]_2, \ldots, [\![s]\!]_{t-1})$
      - Lagrange Interpolation
    - Set $[\![s]\!]_j \coloneqq f\left(x_j\right)$ for $j \geq t$
    - Output $[\![s]\!]_1, [\![s]\!]_2, \ldots, [\![s]\!]_n$

Shamir Secret Sharing

- Uses polynomials over a field $\mathbb{F}$
- **Fact:** Suppose that $\mathrm{p(x)} = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ is a polynomial over a field $|\mathbb{F}| > t$ and let $x_1, \ldots, x_t$ be any set of t distinct points on the field. Then the polynomial $\mathrm{p(x)}$ is uniquely determined by the outputs $\left(\mathrm{p}(x_1), \ldots, p(x_t)\right)$

- View secret $s \in \mathbb{F}$ as a field element
  - Fix distinct field elements $x_0, \ldots, x_n \in \mathbb{F}$
  - $\mathrm{ShareGen(s, t, n)}$
    - Pick $[\![s]\!]_1, [\![s]\!]_2, \ldots, [\![s]\!]_{t-1} \in \mathbb{F}$ uniformly at random
    - Define the polynomial $f(x)$ such that $\left(\mathrm{f}(x_0), \ldots, f(x_{t-1})\right) = (\mathrm{s}, [\![s]\!]_1, [\![s]\!]_2, \ldots, [\![s]\!]_{t-1})$
      - Lagrange Interpolation
    - Set $[\![s]\!]_j \coloneqq f\left(x_j\right)$ for $j \geq t$
    - Output $[\![s]\!]_1, [\![s]\!]_2, \ldots, [\![s]\!]_n$
  - $\mathrm{Recover()}$ uses Lagrange Interpolation to extract polynomial and recover $\mathrm{f}(x_0)$

Binary Secret Sharing Trick

- Alice has shares $[\![t]\!]_1$ and $[\![s]\!]_1$ of secret bits t and s
- Bob has shares $[\![t]\!]_2$ and $[\![s]\!]_2$ of secret bits t and s
- $[\![t]\!]_1 \oplus [\![t]\!]_2 = t$
- $[\![s]\!]_1 \oplus [\![s]\!]_2 = s$

- **Trick 1 Linearity:** $[\![y]\!]_i = [\![t]\!]_i \oplus [\![s]\!]_i$ is a share of $s \oplus t$
$$[\![y]\!]_1 \oplus [\![y]\!]_2 = ([\![t]\!]_1 \oplus [\![s]\!]_1) \oplus ([\![t]\!]_2 \oplus [\![s]\!]_2) = s \oplus t$$
  - Alice can compute $[\![y]\!]_1 = [\![t]\!]_1 \oplus [\![s]\!]_1$ locally
  - Bob can compute $[\![y]\!]_2 = [\![t]\!]_2 \oplus [\![s]\!]_2$ locally

Binary Secret Sharing Trick

- Alice has shares $[\![t]\!]_1$ and $[\![s]\!]_1$ of secret bits t and s
- Bob has shares $[\![t]\!]_2$ and $[\![s]\!]_2$ of secret bits t and s
- $[\![t]\!]_1 \oplus [\![t]\!]_2 = t$
- $[\![s]\!]_1 \oplus [\![s]\!]_2 = s$

- **Trick 2:** Suppose Alice/Bob want to compute shares of $\mathrm{t} \cdot \mathrm{w}$ (w known).
  - Alice can compute $[\![y]\!]_1 = [\![t]\!]_1 \cdot \mathrm{w}$ locally
  - Bob can compute $[\![y]\!]_2 = [\![t]\!]_2 \cdot \mathrm{w}$ locally

$$[\![y]\!]_1 \oplus [\![y]\!]_2 = ([\![t]\!]_1 \cdot \mathrm{w}) \oplus ([\![t]\!]_2 \cdot \mathrm{w}) = w \cdot t$$

Binary Secret Sharing Trick

- Alice has shares $[\![t]\!]_1$ and $[\![s]\!]_1$of secret bits t and s
- Bob has shares $[\![t]\!]_2$ and $[\![s]\!]_2$of secret bits t and s
- $[\![t]\!]_1 \oplus [\![t]\!]_2 = t$
- $[\![s]\!]_1 \oplus [\![s]\!]_2 = s$

- **Combo:** Suppose Alice/Bob want to compute shares of $\mathrm{s} \oplus (\mathrm{t} \cdot \mathrm{w})$ (w known).
  - Alice can compute $[\![y]\!]_1 = [\![s]\!]_1 \oplus ([\![t]\!]_1 \cdot \mathrm{w})$ locally
  - Bob can compute $[\![y]\!]_2 = [\![s]\!]_2 \oplus [\![t]\!]_2 \cdot \mathrm{w}$ locally

$$[\![y]\!]_1 \oplus [\![y]\!]_2 = \left([\![s]\!]_1 \oplus ([\![t]\!]_1 \cdot \mathrm{w})\right) \oplus \left([\![s]\!]_2 \oplus ([\![t]\!]_2 \cdot \mathrm{w})\right) = \begin{cases} s & \text{if } w = 0 \\ \mathrm{s} \oplus \mathrm{t} & \text{otherwise} \end{cases}$$

Distributed Point Function

- Point Function:

$$f_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

- Alice/Bob each get DPF keys K_0 and K_1
  - Alice can use K_0 to compute a share $[\![s_x]\!]_0$ of $f_{\alpha,\beta}(x)$ for any input x
  - Bob can use K_1 to compute a share $[\![s_x]\!]_1$ of $f_{\alpha,\beta}(x)$ for any input x
- **Correctness:** for all inputs x

$$[\![s_x]\!]_0 \oplus [\![s_x]\!]_1 = f_{\alpha,\beta}(x)$$

Distributed Point Function

- Point Function:

$$f_{\alpha,\beta}(x) \coloneqq \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

- Alice/Bob each get DPF keys K_0 and K_1
  - Alice can use K_0 to compute a share $[\![s_x]\!]_0$ of $f_{\alpha,\beta}(x)$ for any input x
  - Bob can use K_1 to compute a share $[\![s_x]\!]_1$ of $f_{\alpha,\beta}(x)$ for any input x
- **Privacy:** Alice/Bob should not learn anything about the secrets α, β

Distributed Point Function

- Point Function:

$$f_{\alpha,\beta}(x) \coloneqq \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

- Alice/Bob each get DPF keys K_0 and K_1
  - Alice can use K_0 to compute a share $[\![s_x]\!]_0$ of $f_{\alpha,\beta}(x)$ for any input x
  - Bob can use K_1 to compute a share $[\![s_x]\!]_1$ of $f_{\alpha,\beta}(x)$ for any input x
- **Solution 1:** Generate shares $[\![s_\alpha]\!]_0 \oplus [\![s_\alpha]\!]_1 = f_{\alpha,\beta}(\alpha)$ and set $K_i = (\alpha, [\![s_\alpha]\!]_i)$
- Violates privacy

Distributed Point Function

- Point Function:

$$f_{\alpha,\beta}(x) \coloneqq \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

- Alice/Bob each get DPF keys K_0 and K_1
  - Alice can use K_0 to compute a share $[\![s_x]\!]_0$ of $f_{\alpha,\beta}(x)$ for any input x
  - Bob can use K_1 to compute a share $[\![s_x]\!]_1$ of $f_{\alpha,\beta}(x)$ for any input x
- **Solution 2:** Generate shares $[\![s_x]\!]_0 \oplus [\![s_x]\!]_1 = f_{\alpha,\beta}(x)$ for each input x and set $K_i = \{[\![s_x]\!]_i\}_{x \in \{0,1\}^n}$
- Private/Correct ☺
- **Problem:** Exponentially large keys! ☹

GGM Based Distributed Point Function

- **Attempt 1:** Alice/Bob both get K which is the root of a GGM tree.
  - Alice and Bob can both evaluate $F_K(x)$
  - Alice and Bob obtain shares of $F_K(x) \oplus F_K(x) = 0$
  - Incorrect when $x = \alpha$

- **Attempt 2:** Puncture key at $x = \alpha$
  - Give Alice/Bob punctured Key $K[\alpha]$
  - Generate shares $[\![s_\alpha]\!]_0 \oplus [\![s_\alpha]\!]_1 = f_{\alpha,\beta}(\alpha) = \beta$
  - Give Alice/Bob the shares $[\![s_\alpha]\!]_0$ and $[\![s_\alpha]\!]_1$ respectively
  - Correct ☺
  - Hides β ☺
  - Does not hide α ☹

$$\mathbf{G}(\mathbf{x}) := \underbrace{\mathbf{G_0}(\mathbf{x})}_{\lambda\text{-bits}} \,||\, \underbrace{\mathbf{G_1}(\mathbf{x})\mathbf{s_1}\mathbf{t_1}}_{\lambda\text{-bits}}$$

λ bits

$$f_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

GGM Based Distributed Point Function

- **Attempt 3 (Obfuscation):**
  - Pick PPRF keys K
  - Define DPF keys

$$K_0 = iO(C_0), K_1 = iO(C_1)$$

Where

$$C_0(x) \coloneqq F_K(x)$$

$$C_1(x) \coloneqq \begin{cases} \beta \oplus F_K(x) & if x = \alpha \\ F_K(x) & otherwise \end{cases}$$

$$G(x) := \underbrace{G_0(x)}_{\lambda\text{-bits}} \,||\, \underbrace{G_1(x)s_1t_1}_{\lambda\text{-bits}}$$

λ bits

$$f_{\alpha,\beta}(x) \coloneqq \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

Advantages:
Correct! ☺

$$K_0(\alpha) \oplus K_1(\alpha) = F_K(\alpha) \oplus \beta \oplus F_K(\alpha) = \beta$$

If $x \neq \alpha$

$$K_0(x) \oplus K_1(x) = F_K(x) \oplus F_K(\alpha) = 0$$

GGM Based Distributed Point Function

- **Attempt 3 (Obfuscation):**
  - Pick PPRF key K
  - Define DPF keys

$$K_0 = iO(C_0), K_1 = iO(C_1)$$

Where

$$C_0(x) \coloneqq F_K(x)$$

$$C_1(x) \coloneqq \begin{cases} \beta \oplus F_K(x) & if x = \alpha \\ F_K(x) & otherwise \end{cases}$$

$$\underset{\lambda \text{ bits}}{G(x)} \coloneqq \overbrace{G_0(x)}^{\lambda\text{ -bits}} \,||\, \overbrace{G_1(x)s_1t_1}^{\lambda\text{-bits}}$$

$$f_{\alpha,\beta}(x) \coloneqq \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

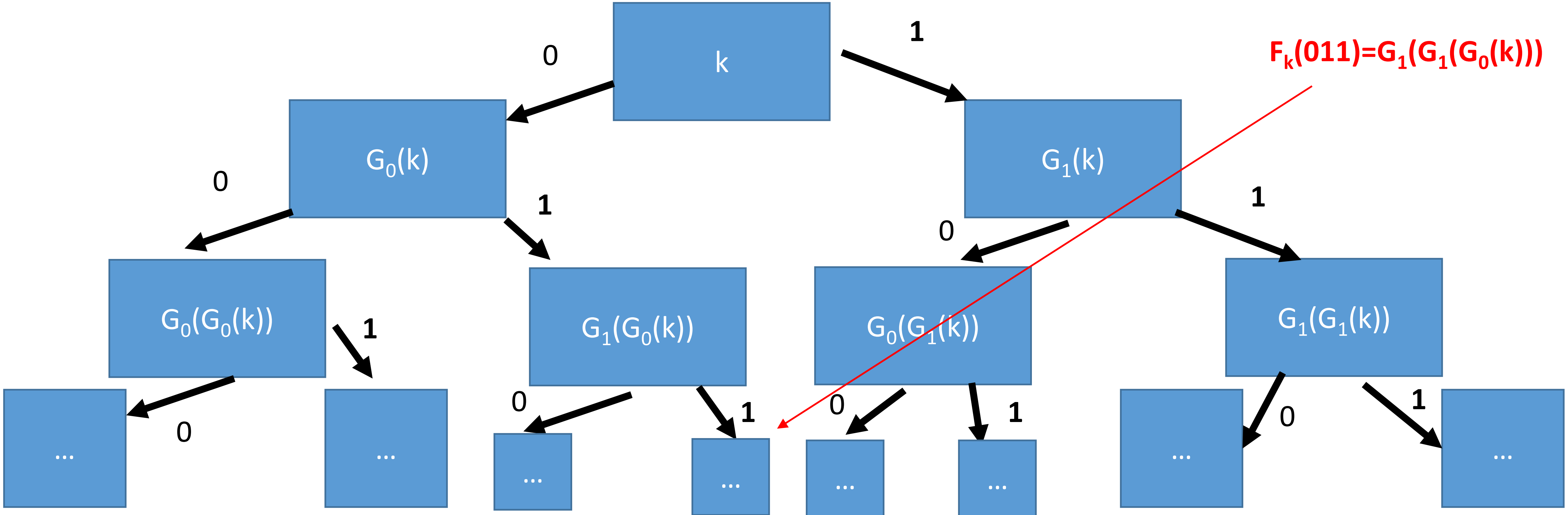
Advantages:
  Security?
  See homework 3 ☺

Disadvantage:
 Highly impractical!

Recap: PPRFs from PRGs

$$G(x) := \overbrace{G_0(x)}^{\lambda\text{-bits}} \,||\, \overbrace{G_1(x)}^{\lambda\text{-bits}}$$

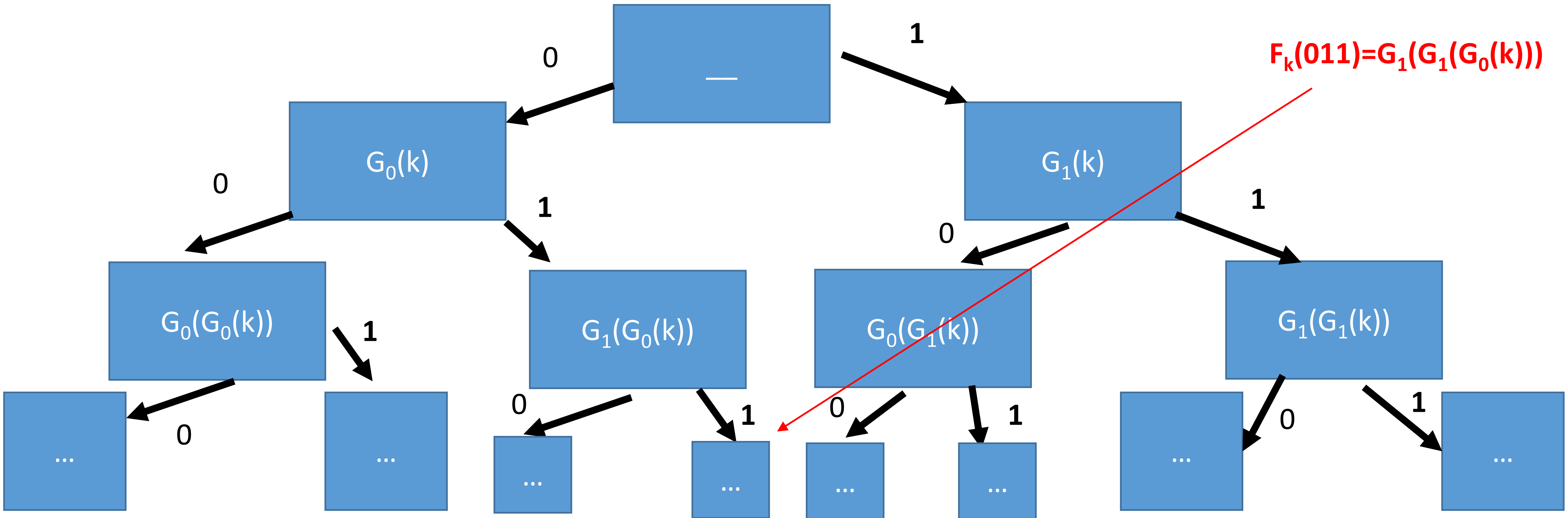
GGM Puncturable PRF Construction



Recap: PPRFs from PRGs

$$G(x) := \overbrace{G_0(x)}^{\lambda\text{-bits}} \,||\, \overbrace{G_1(x)}^{\lambda\text{-bits}}$$

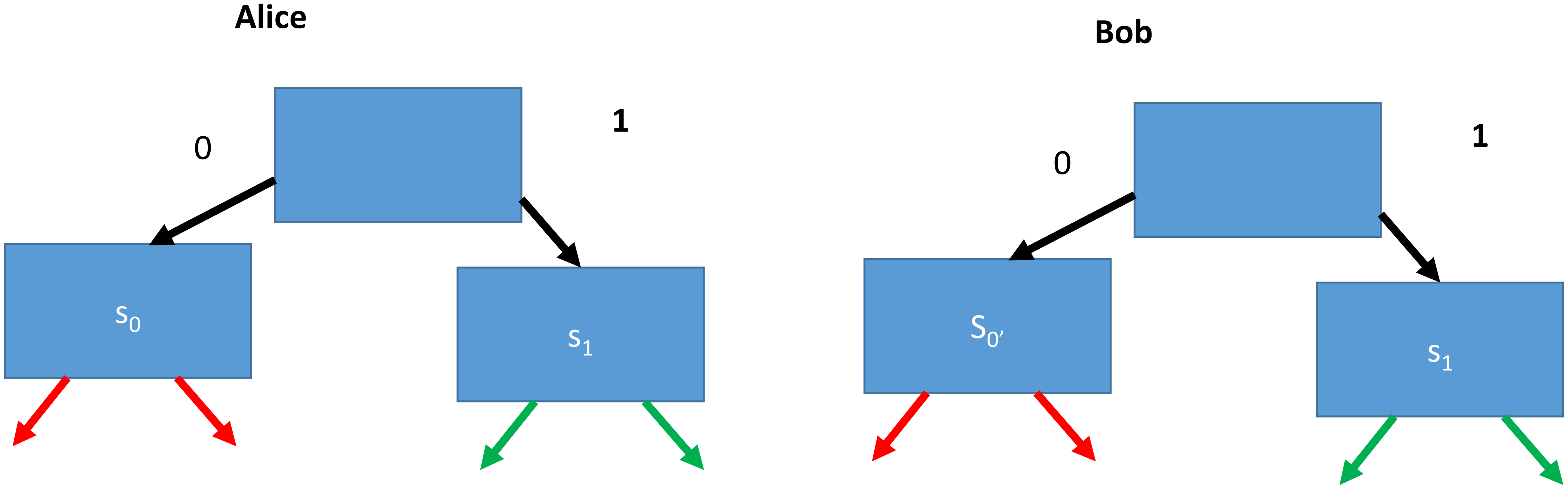
Could start with keys $G_0(k)$ and $G_1(k)$ instead of k



DPF Idea

$$G(x) := \overbrace{G_0(x)}^{\lambda\text{-bits}} \,||\, \overbrace{G_1(x)}^{\lambda\text{-bits}}$$

Suppose first bit $\alpha_1 = 0$

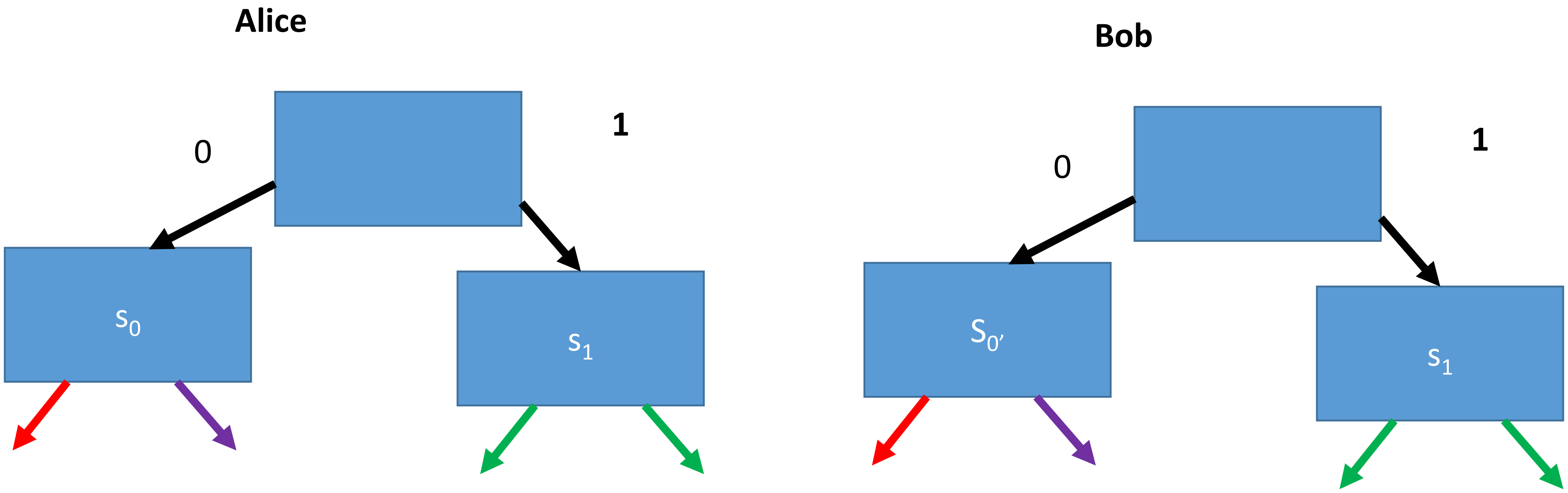


Alice/Bob compute same thing on green paths/different things on red paths

DPF Idea

$$G(x) := \overbrace{G_0(x)}^{\lambda\text{-bits}} \,||\, \overbrace{G_1(x)}^{\lambda\text{-bits}}$$

Suppose first bit $\alpha_1 = 0$

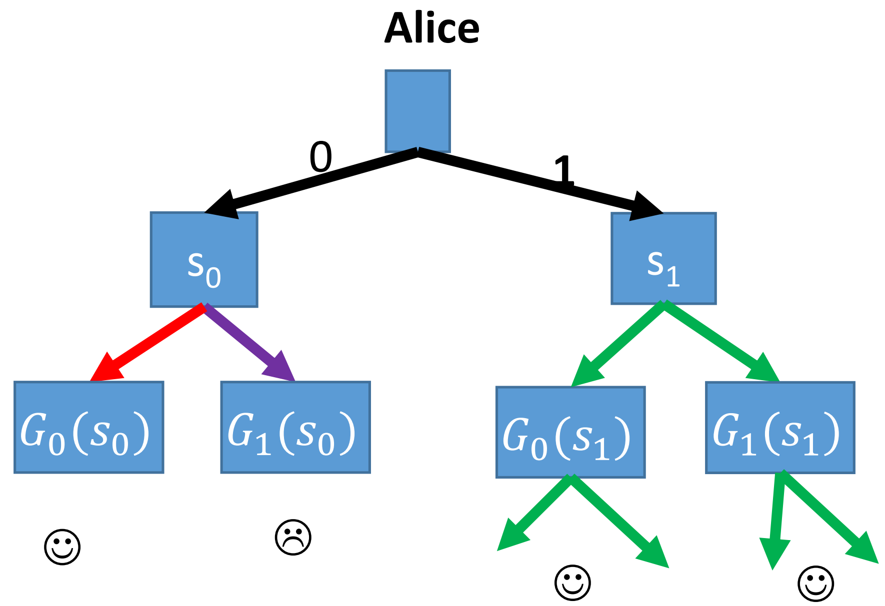


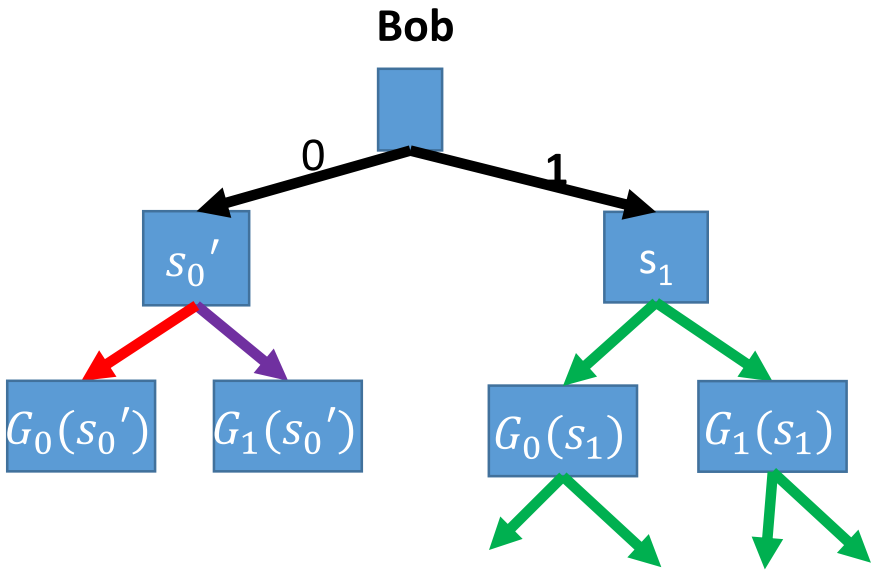
If $\alpha_2 = 0$ we want purple path to converge

DPF: Fix Attempt 1

$$\mathbf{G(x) := \underbrace{G_0(x)}_{\lambda\text{-bits}} \,||\, \underbrace{G_1(x)}_{\lambda\text{-bits}}}$$

Suppose first bit $\alpha_1 = 0$ and second bit is $\alpha_2 = 0$

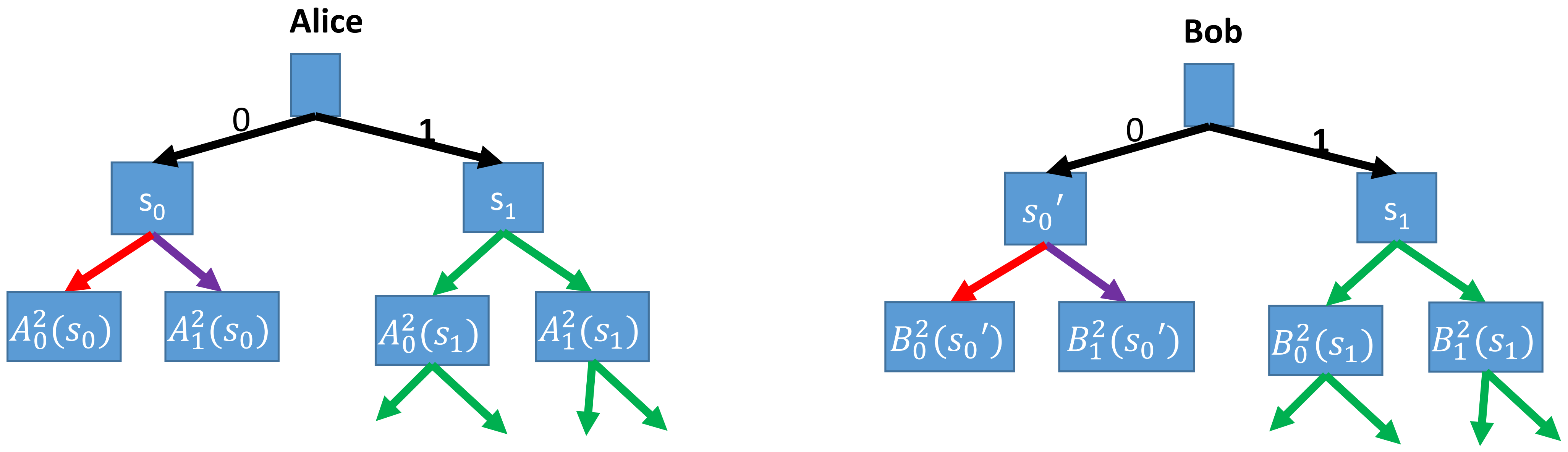




DPF: Fix Attempt 1

$$\mathbf{G(x) := \underbrace{G_0(x)}_{\lambda\text{-bits}} \,||\, \underbrace{G_1(x)}_{\lambda\text{-bits}}}$$

Suppose first bit $\alpha_1 = 0$ and second bit is $\alpha_2 = 0$



Level 2: Pick random strings R_0^B, R_1^B, and set $R_1^A = (G_1(s_0) \oplus G_1(s_0{}')) \oplus R_0^B$ and $R_0^A = R_0^B$

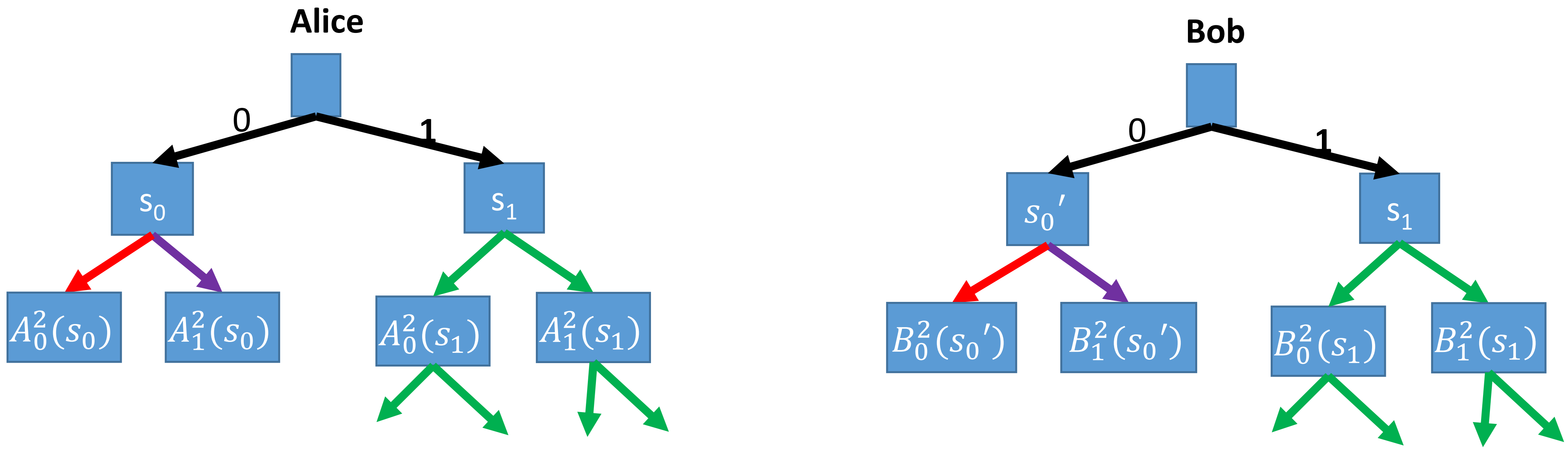
Bob uses function $B_0^2(x) = G_0(x) \oplus R_0^B$ and $B_1^2(x) = G_1(x) \oplus R_1^B$

Alice defines functions $A_0^2(x) = G_0(x) \oplus R_0^A$ and $A_1^2(x) = G_1(x) \oplus R_1^A$

DPF: Fix Attempt 1

$$\mathbf{G(x)} := \overbrace{\mathbf{G_0(x)}}^{\lambda\text{-bits}} \mid\mid \overbrace{\mathbf{G_1(x)}}^{\lambda\text{-bits}}$$

Suppose first bit $\alpha_1 = 0$ and second bit is $\alpha_2 = 0$



Level 2: Pick random strings R_0^B, R_1^B, and set $R_1^A = (G_1(s_0) \oplus G_1(s_0{}')) \oplus R_0^B$ and $R_0^A = R_0^B$

Warning: If we make all random strings public then Alice/Bob learn $\alpha_2 = 0$

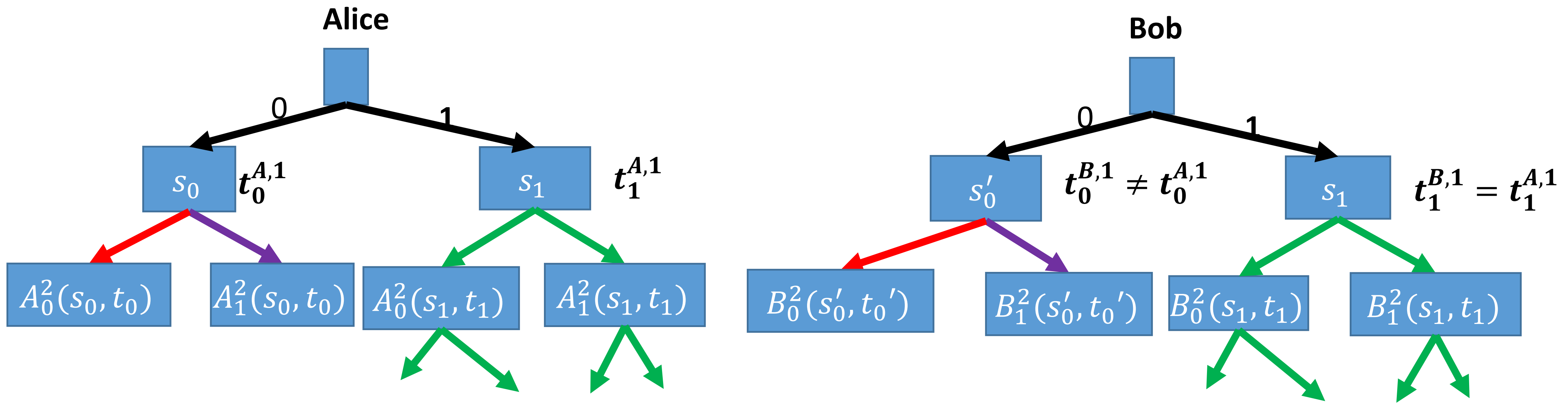
Solution: Some random strings are public; some are given only to Alice (resp. Bob).

DPF: Control Bits

$$\mathbf{G(x) := \overbrace{G_0(x)}^{\lambda+1\text{ -bits}} \,||\, \overbrace{G_1(x)}^{\lambda+1\text{ -bits}}}$$

λ –bit input

Suppose first bit $\alpha_1 = 0$ and second bit is $\alpha_2 = 0$



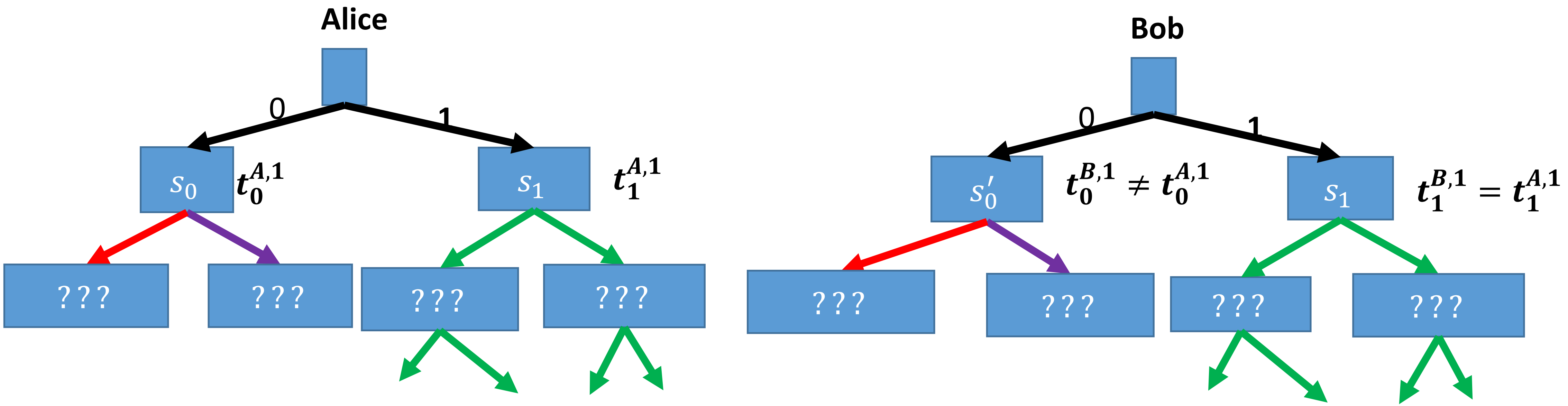
At each level i Alice (resp. Bob) will have secret control bits $t_0^{A,i}$ and $t_1^{A,i}$ (resp. $t_0^{B,i}$ and $t_1^{B,i}$) which are part of the private key

Guarantee: $t_{1-\alpha_1}^{B,1} = t_{1-\alpha_1}^{A,1}$ and $t_{\alpha_1}^{B,1} = t_{\alpha_1}^{A,1}$

DPF: Invariants

$$G(\underbrace{x}_{\lambda\text{–bit input}}) := \overbrace{G_0(x)}^{\lambda+1\text{ -bits}} \;||\; \overbrace{G_1(x)}^{\lambda+1\text{ -bits}}$$

Suppose first bit $\alpha_1 = 0$ and second bit is $\alpha_2 = 0$



Invariant (Control Bits): At each node *on the red path* Alice/Bob can locally compute secret shares of [1] and at each node off this path Alice/Bob have secret shares of [0].

Invariant: At each node off the path Alice/Bob can locally compute secret shares of 0^λ and at each node on the red path Alice/Bob have shares of a pseudorandom λ –bit string R

Conditional Correction Gadget

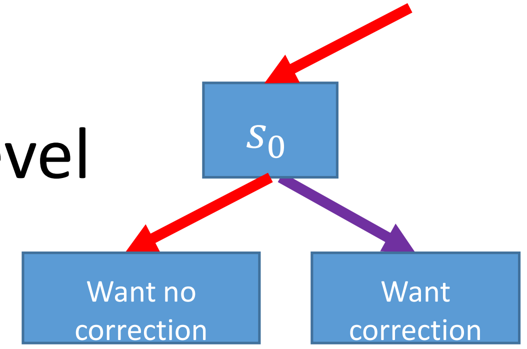
- Alice has R_0 and b_0 and
- Bob has $R_1 = R \oplus R_0$ and $b_1 = b \oplus b_0$
- Public Correction Factor Δ

- Bob computes $R_1' = R_1 \oplus (b_1 \Delta)$ and Alice computes $R_0' = R_0 \oplus (b_0 \Delta)$

$$R_1' \oplus R_0' = R \oplus (b_1 \Delta) \oplus (b_0 \Delta) = R \oplus (b\Delta)$$

- Thus, Alice/Bob can locally obtain shares of $R \oplus (b\Delta)$
  - If b=0 (e.g., already off path) then net effect is that no correction is applied
    - Already off path (R=0,b=0) ➔ $R \oplus (b\Delta) = 0$ (Secret shares of 0!)
  - If b=1 (e.g., was still on path) then net effect is that correction is applied
    - If $\Delta =$ R,b=1 ➔ $R \oplus (b\Delta) = 0$ (Secret shares of 0 again!)

Conditional Correction Gadget: Attempt 1



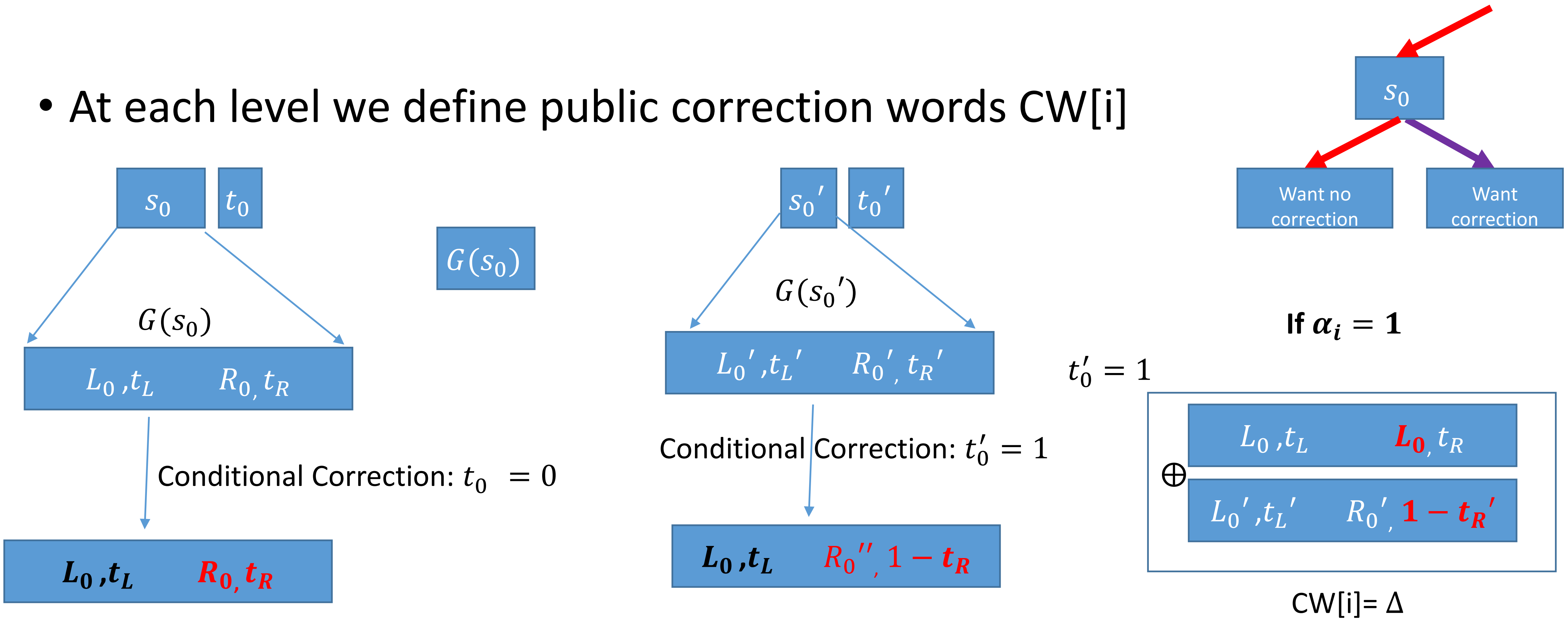
- Define conditional correction factors Δ_0^i and Δ_1^i for each level
  - $\Delta_{\alpha_i}^i = 0$ (stay on path ➔ no correction)
  - $\Delta_{1-\alpha_i}^i = R$ (leave path ➔ want to apply correction)

- Alice/Bob can apply correction factor $\Delta_{x_i}^i$

- Problem?
  - Alice/Bob can figure out α_i from the value Δ_0^i and Δ_1^i!
  - Can we make $\Delta_{\alpha_i}^i$ ``look random”?

Correction Words (On Path)

- At each level we define public correction words CW[i]

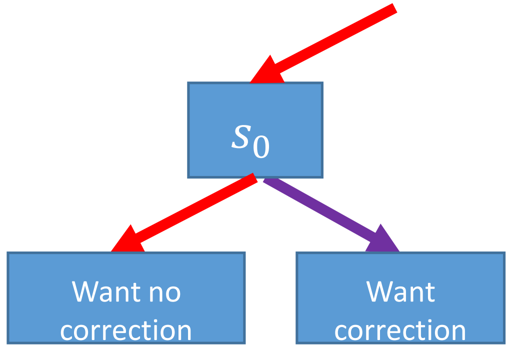


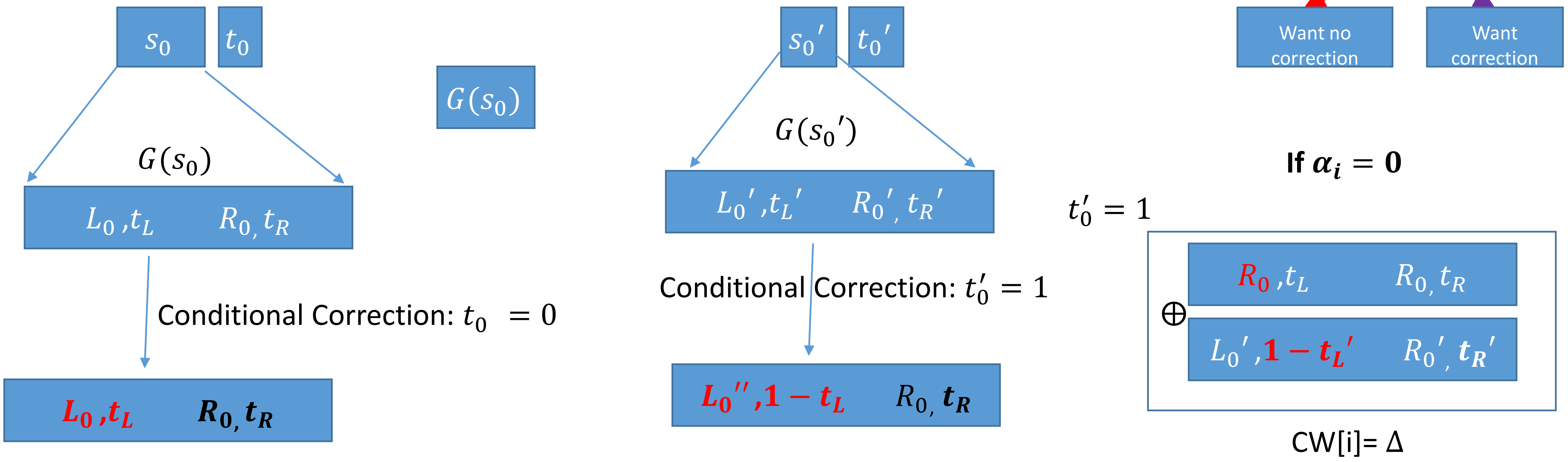
Invariants: If $x_i = 0$ (**exit path**) ➔ **Alice/Bob have shares of zero i.e.,** $L_0 \oplus L_0 = 0$ **and** $t_L \oplus t_L = 0$

If $x_i = 1$ (**stay on path**) ➔ **Alice/Bob have shares of pseudorandom** $R_0 \oplus R_0''$ and $t_R + (1 - t_R) = 1$

Correction Words (On Path)

- At each level we define public correction words CW[i]



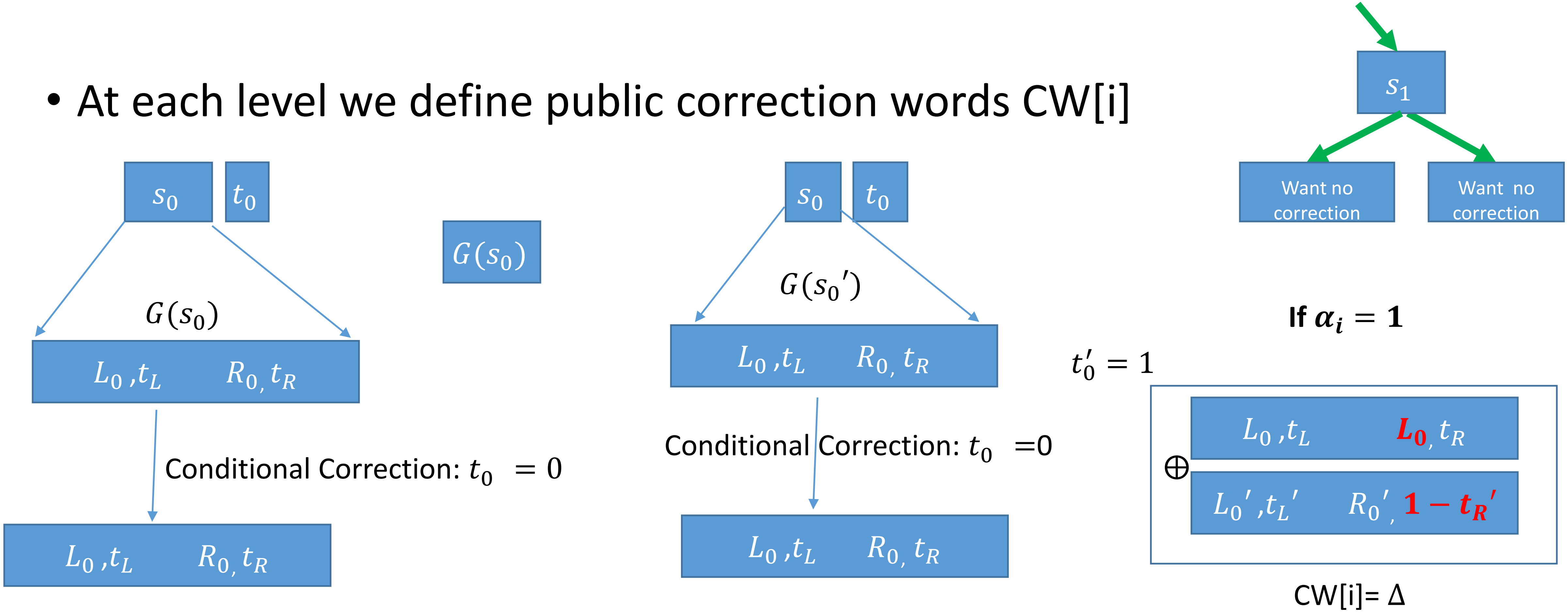


Invariants: If $x_i = 1$ (**exit path**) ➔ **Alice/Bob have shares of zero i.e.,** $R_0 \oplus R_0 = 0$ **and** $t_R \oplus t_R = 0$

If $x_i = 0$ (**stay on path**) ➔ **Alice/Bob have shares of pseudorandom** $L_0 \oplus L_0''$ and $t_L + (1 - t_L) = 1$

Correction Words (Already off path)

- At each level we define public correction words CW[i]



Invariants: If $x_i = 0$ **(remain off path)** ➔ **Alice/Bob have shares of zero i.e.,** $L_0 \oplus L_0 = 0$ **and** $t_L \oplus t_L = 0$

If $x_i = 1$ **(remain off path)** ➔ **Alice/Bob have shares of pseudorandom** $R_0 \oplus R_0$ and $t_R \oplus t_R = 0$

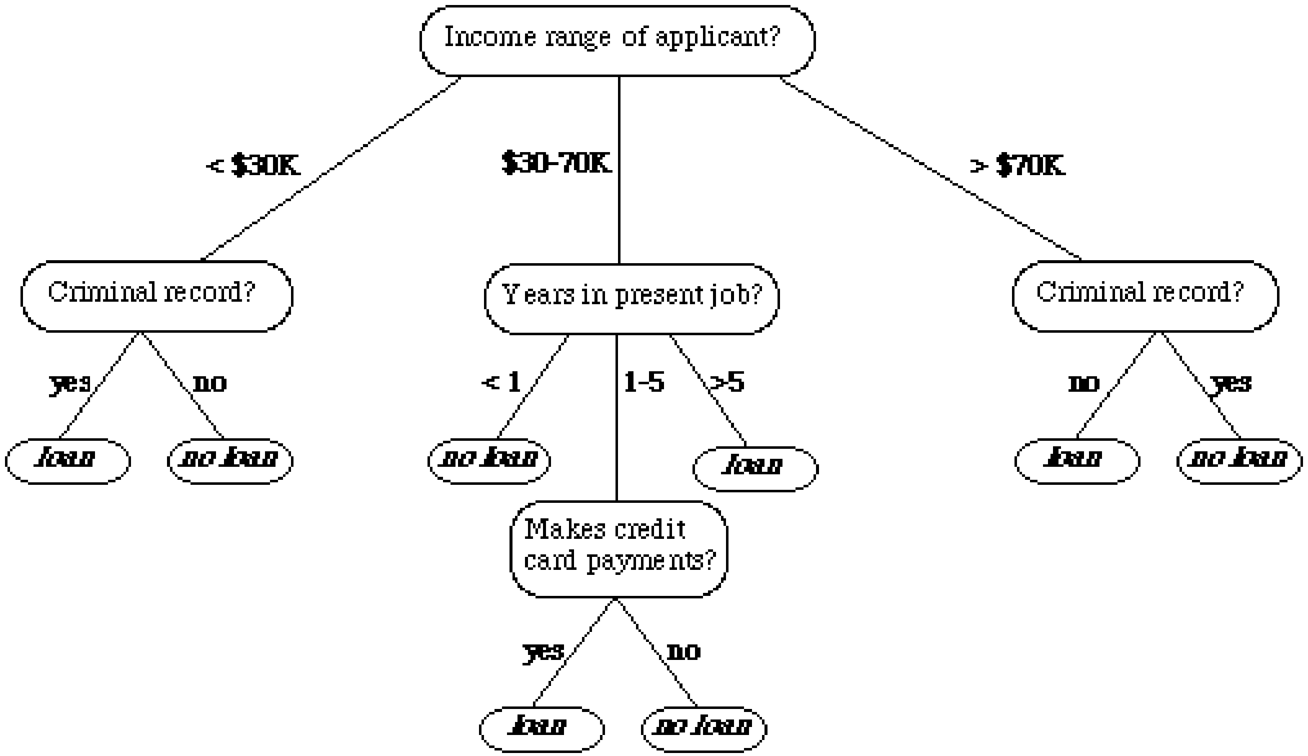
Distributed Point Function: Complexity

- Function Share/Key Size
  - PRG Seed (λ bits)
  - Correction Word at Each Level: $O(\lambda n)$ bits total

- Key Generation (Time)
  - n PRG evaluations (plus a few XORs)
- Evaluation:
  - n PRG evaluations (plus a few XORs)
  - Essentially the same as

Other Examples of Functional Secret Sharing

- FSS for Decision Trees
  - Applications to Machine Learning



Fully Homomorphic Encryption (FHE)

- Idea: Alice sends Bob $Enc_{PK_A}(x_1), \dots, Enc_{PK_A}(x_n)$

$$Enc_{PK_A}(x_i) + Enc_{PK_A}(x_j) = Enc_{PK_A}(x_i + x_j)$$

and

$$Enc_{PK_A}(x_i) \times Enc_{PK_A}(x_j) = Enc_{PK_A}(x_i \times x_j)$$

- Bob cannot decrypt messages, but given a circuit C can compute

$$Enc_{PK_A}(C(x_1, \dots, x_n))$$

- Proposed Application: Export confidential computation to cloud

https://simons.berkeley.edu/talks/shai-halevi-2015-05-18a (Lecture by Shai Halevi)

Fully Homomorphic Encryption (FHE)

- Idea: Alice sends Bob $Enc_{PK_A}(x_1), \ldots, Enc_{PK_A}(x_n)$
- Bob cannot decrypt messages, but given a circuit C can compute

$$Enc_{PK_A}\big(C(x_1, \ldots, x_n)\big)$$

- We now have candidate constructions!
  - Encryption/Decryption are polynomial time
  - …but expensive in practice.
  - Proved to be CPA-Secure under plausible assumptions

- Remark 1: Partially Homomorphic Encryption schemes cannot be CCA-Secure. Why not?

https://simons.berkeley.edu/talks/shai-halevi-2015-05-18a (Lecture by Shai Halevi)

Partially Homomorphic Encryption

- Plain RSA is multiplicatively homomorphic

$$Enc_{PK_A}(x_i) \times Enc_{PK_A}(x_j) = Enc_{PK_A}(x_i \times x_j)$$

- But not additively homomorphic

- Pallier Cryptosystem

$$Enc_{PK_A}(x_i) \times Enc_{PK_A}(x_j) = Enc_{PK_A}(x_i + x_j)$$

$$\left(Enc_{PK_A}(x_i)\right)^k = Enc_{PK_A}(k \times x_j)$$

- Not same as FHE, but still useful in multiparty computation

https://en.wikipedia.org/wiki/Paillier_cryptosystem

Partially Homomorphic Encryption

- **Secret Key:** Large (prime) number p.
- **Public Key:** $\mathrm{N} = \mathrm{pq}$ and $x_i = pq_i + 2r_i + 1$ for each $i \le t$ where $r_i \ll p$

- Encrypting a Bit b:
  - Select Random Subset: $S \subset [t]$ and random $r \ll p$
  - Return $c = b + 2r + \sum_{i \in S} x_i \ \ mod\ N = p\sum_{i \in S} q_i + 2(r + \sum_{i \in S} r_i) + b\ mod\ N$
- Decrypting a ciphertext:
  - *As long as* $2(r + \sum_{i \in S} r_i) < p$
  - $(c\ mod\ p) mod\ 2 = (2(r + \sum_{i \in S} r_i) + b\ ) \ mod\ 2 = b$

Partially Homomorphic Encryption

- Encrypting a Bit b:
  - Select Random Subset: $S \subset [t]$ and random $r \ll p$
  - Return $c = b + 2r + \sum_{i \in S} x_i \ mod\ N = p \sum_{i \in S} q_i + 2(r + \sum_{i \in S} r_i) + b$
- **Adding two ciphertexts**

$$c + c' = p\left(\sum_{i \in S} q_i + \sum_{i \in S'} q_i\right) + 2\underbrace{\left(r + r' + \sum_{i \in S} r_i + \sum_{i \in S'} r_i\right)}_{\textbf{Noise increases a bit}} + b + b'$$

Partially Homomorphic Encryption

- Encrypting a Bit b:
  - Select Random Subset: $S \subset [t]$ and random $r \ll p$
  - Return $c = b + 2r + \sum_{i \in S} x_i \mod N = p \sum_{i \in S} q_i + 2(r + \sum_{i \in S} r_i) + b$
- **Multiply two ciphertexts**

$$cc' = p\left(\sum_{i \in S} q_i \sum_{i \in S'} q_i + \sum_{i \in S} q_i \sum_{i \in S'} r_i + \cdots\right) +$$

$$\underbrace{4\left(\left(\boldsymbol{r} + \sum_{i \in S} \boldsymbol{r}_i\right)\left(\boldsymbol{r}' + \sum_{i \in S'} \boldsymbol{r}_i\right)\right) + 2b\left(r + \sum_{i \in S'} r_i\right) + 2b'\left(r + \sum_{i \in S} r_i\right)}_{\textbf{Noise increases a bit more (multiplicative)}} + bb'$$

Bootstrapping (Gentry 2009)

- Transform Partially Homomorphic Encryption Scheme into Fully Homomorphic Encryption Scheme
- **Key Idea:**
  - Maintain two public keys pk_1 and pk_2 for partially homomorphic encryption
    - Also, encrypt sk_1 using pk_2 and encrypt sk_2 under pk_1
    - The ciphertexts are included in the public key
  - Run homomorphic evaluation using pk_1 until the noise gets to be too large
  - Let $c_1,\ldots,c_k$ be intermediate ciphertext(s) (under key pk_1)
  - Encrypt $c_1,\ldots,c_k$ bit by bit under (under key pk_2)
  - Then evaluate the decryption circuit homorphically (under key pk_2)
  - **Challenge:** Need to make sure that decryption circuit is shallow enough to evaluate…
- Expensive, but there are tricks to reduce the running time

Fully Homomorphic Encryption Resources

- Implementation: https://github.com/shaih/HElib
- Tutorial: https://www.youtube.com/watch?v=jIWOR2bGC7c

Thanks for Listening