The hybrid argument

Indistinguishability of probability distributions

- For each $\eta \in \mathbb{N}$ let D^0_η and D^1_η be probability distributions over bit-strings.
- The families of probability distributions $D^0 = \{D^0_\eta\}_{\eta\in\mathbb{N}}$ and $D^1 = \{D^1_\eta\}_{\eta\in\mathbb{N}}$ are indistinguishable if
  - for any adversary $\mathcal{A}$
    - The running time of $\mathcal{A}(\eta, \cdot)$ must be polynomial in η
  - the difference of probabilities

$$\Pr[\mathcal{A}(\eta, x) = 1 \,|\, x \leftarrow D^0_\eta] - \Pr[\mathcal{A}(\eta, x) = 1 \,|\, x \leftarrow D^1_\eta]$$

    is a negligible function of η.
- Denote $D^0 \approx D^1$.

Writing code

```
interface SingleEnv {
    bitstring getX();
}
```

```
interface SingleAdv {
    bit guess(SingleEnv envir);
}
```

```
class SingleInd_{D^0,D^1} implements SingleEnv {
    private bitstring x;

    SingleInd_{D^0,D^1}(bit b_0) {
        x ← D^{b_0};
    }

    bitstring getX() {
        return x;
    }
}
```

We have (t, ε)-indistinguishability, if for all adversaries $\mathcal{A}$ that run in time t and implement SingleAdv,

$$\left|\Pr[b \in_R \{0,1\};\ \mathcal{A}.\mathit{guess}(\textbf{new}\ \mathsf{SingleInd}_{D^0,D^1}(b)) = b] - \frac{1}{2}\right| \leq \varepsilon \ .$$

With security parameter

```
interface SingleEnv {
    bitstring getX();
}
interface SingleAdv {
    bit guess(int η, SingleEnv envir);
}
class SingleInd_{D^0,D^1} implements SingleEnv {
    private bitstring x;

    SingleInd_{D^0,D^1}(int η, bit b_0) {
        x ← D_η^{b_0};
    }
    bitstring getX() {
        return x;
    }
}
```

We have (uniform polynomial) indistinguishability, if for all adversaries $\mathcal{A}$ that run in polynomial time (wrt. its first parameter) and implement SingleAdv,

$$\left|\Pr[b \in_R \{0,1\};\ \mathcal{A}.\mathit{guess}(\textbf{new}\ \mathsf{SingleInd}_{D^0,D^1}(\eta,\ b)) = b] - \frac{1}{2}\right|$$

is a negligible function of η.

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that $\mathsf{SingleInd}_{D^0,D^2}(\eta, 0)$ may be replaced with $\mathsf{SingleInd}_{D^0,D^2}(\eta, 1)$.

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

class SingleInd$_{D^0,D^2}$ **implements** SingleEnv $\{$

  private bitstring x;

  SingleInd$_{D^0,D^2}$(**int** η, **bit** b_0) $\{$

    $x \leftarrow D_\eta^{2 \cdot b_0}$;

  $\}$

  bitstring *getX*() $\{$

    return x;

  $\}$

$\}$

Call **new** SingleInd$_{D^0,D^2}(\eta, 0)$

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that $\mathsf{SingleInd}_{D^0,D^2}(\eta, 0)$ may be replaced with $\mathsf{SingleInd}_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private bitstring x;                      bitstring getX() {
                                                  return x;
    SingleInd_{D^0,D^2}(int η, bit b_0) {     }
        x ← D_η^{2·b_0};                  }
    }
Call new SingleInd_{D^0,D^2}(η, 0)
```

Propagate copies

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private bitstring x;

    SingleInd_{D^0,D^2}(int η, bit b_0) {
        x ← D^0_η;
    }
                                        bitstring getX() {
                                            return x;
                                        }
                                    }
Call new SingleInd_{D^0,D^2}(η, 0)
```

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that $\text{SingleInd}_{D^0,D^2}(\eta, 0)$ may be replaced with $\text{SingleInd}_{D^0,D^2}(\eta, 1)$.

class SingleInd$_{D^0,D^2}$ **implements** SingleEnv $\{$
    private bitstring x;

    SingleInd$_{D^0,D^2}$(**int** η, **bit** b_0) $\{$
        $x \leftarrow D^0_\eta$;
    $\}$

    bitstring *getX*() $\{$
        return x;
    $\}$
$\}$

Call **new** SingleInd$_{D^0,D^2}(\eta, 0)$

Keep x inside $\text{SingleInd}_{D^0,D^1}(\eta, 0)$

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private SingleEnv e;

    SingleInd_{D^0,D^2}(int η, bit b_0) {
        e :=new SingleInd_{D^0,D^1}(η, 0);
    }
                                              bitstring getX() {
                                                  return e.getX();
                                              }
                                          }
Call new SingleInd_{D^0,D^2}(η, 0)
```

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private SingleEnv e;
    SingleInd_{D^0,D^2}(int η, bit b_0) {
        e :=new SingleInd_{D^0,D^1}(η, 0);
    }
    bitstring getX() {
        return e.getX();
    }
}
Call new SingleInd_{D^0,D^2}(η, 0)
```

Use $D^0 \approx D^1$

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private SingleEnv e;

    SingleInd_{D^0,D^2}(int η, bit b_0) {
        e :=new SingleInd_{D^0,D^1}(η, 1);
    }
                                    bitstring getX() {
                                        return e.getX();
                                    }
                                  }
Call new SingleInd_{D^0,D^2}(η, 0)
```

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private SingleEnv e;

    SingleInd_{D^0,D^2}(int η, bit b_0) {
        e := new SingleInd_{D^0,D^1}(η, 1);
    }
                                              bitstring getX() {
                                                  return e.getX();
                                              }
}
Call new SingleInd_{D^0,D^2}(η, 0)
```

Take x out of SingleInd$_{D^0,D^1}(\eta, 1)$

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that $\mathsf{SingleInd}_{D^0,D^2}(\eta, 0)$ may be replaced with $\mathsf{SingleInd}_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private bitstring x;

    SingleInd_{D^0,D^2}(int η, bit b_0) {
        x ← D^1_η;
    }
Call new SingleInd_{D^0,D^2}(η, 0)

    bitstring getX() {
        return x;
    }
}
```

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private bitstring x;

    SingleInd_{D^0,D^2}(int η, bit b_0) {
        x ← D^1_η;
    }

    bitstring getX() {
        return x;
    }
}
```

Call **new** SingleInd$_{D^0,D^2}(\eta, 0)$

Keep x inside SingleInd$_{D^1,D^2}(\eta, 0)$

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private SingleEnv e;

    SingleInd_{D^0,D^2}(int η, bit b_0) {
        e := new SingleInd_{D^1,D^2}(η, 0);
    }
                                            bitstring getX() {
                                                return e.getX();
                                            }
                                         }
Call new SingleInd_{D^0,D^2}(η, 0)
```

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that $\mathsf{SingleInd}_{D^0,D^2}(\eta, 0)$ may be replaced with $\mathsf{SingleInd}_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private SingleEnv e;

    SingleInd_{D^0,D^2}(int η, bit b_0) {
        e :=new SingleInd_{D^1,D^2}(η, 0);
    }
                                          bitstring getX() {
                                              return e.getX();
                                          }
                                      }
Call new SingleInd_{D^0,D^2}(η, 0)
```

Use $D^1 \approx D^2$

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

```
class SingleInd_{D^0,D^2} implements SingleEnv {
    private SingleEnv e;

    SingleInd_{D^0,D^2}(int η, bit b_0) {
        e := new SingleInd_{D^1,D^2}(η, 1);
    }
                                        bitstring getX() {
                                            return e.getX();
                                        }
                                      }
Call new SingleInd_{D^0,D^2}(η, 0)
```

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

class SingleInd$_{D^0,D^2}$ **implements** SingleEnv $\{$
  private SingleEnv e;

  SingleInd$_{D^0,D^2}$(**int** η, **bit** b_0) $\{$
    e :=**new** SingleInd$_{D^1,D^2}(\eta, 1)$;
  $\}$

  bitstring *getX*() $\{$
    return e.*getX*();
  $\}$
$\}$

Call **new** SingleInd$_{D^0,D^2}(\eta, 0)$

Take x out of SingleInd$_{D^1,D^2}(\eta, 1)$

Transitivity

Theorem. If $D^0 \approx D^1$ and $D^1 \approx D^2$, then $D^0 \approx D^2$.

Code-based proof: We have to show that SingleInd$_{D^0,D^2}(\eta, 0)$ may be replaced with SingleInd$_{D^0,D^2}(\eta, 1)$.

class SingleInd$_{D^0,D^2}$ **implements** SingleEnv $\{$
  private bitstring x;

  SingleInd$_{D^0,D^2}$(**int** η, **bit** b_0) $\{$
    $x \leftarrow D^2_\eta$;
  $\}$

  bitstring *getX*() $\{$
    return x;
  $\}$
$\}$

Call **new** SingleInd$_{D^0,D^2}(\eta, 0)$

This is what you get calling **new** SingleInd$_{D^0,D^2}(\eta, 1)$ □

“classical” proof

- Suppose that $D^0 \not\approx D^2$.
- Let $\mathcal{A}$ be a polynomial-time adversary such that $\mathcal{A}$ can distinguish D^0 and D^2 with non-negligible advantage.
- For $i \in \{0, 1, 2\}$, let

$$p_\eta^i = \Pr[\mathcal{A}(\eta, x) = 1 \mid x \leftarrow D_\eta^i]$$

- There is a polynomial q, such that for infinitely many η, $|p_\eta^0 - p_\eta^2| \geq q(\eta)$.
- For any such η, either $|p_\eta^0 - p_\eta^1| \geq q(\eta)/2$ or $|p_\eta^1 - p_\eta^2| \geq q(\eta)/2$.
- Either $|p_\eta^0 - p_\eta^1| \geq q(\eta)/2$ holds for infinitely many η, or $|p_\eta^1 - p_\eta^2| \geq q(\eta)/2$ holds for infinitely many η.
- $\mathcal{A}$ distinguishes either D^0 and D^1, or D^1 and D^2. $\square$

Independent components

- Let D^0, D^1, E be families of probability distributions.
- Define the probability distribution F^i_η by
  1. Let $x \leftarrow D^i_\eta$.
  2. Let $y \leftarrow E_\eta$.
  3. Output (x, y).
- E is polynomial-time constructible if there is a polynomial-time algorithm $\mathcal{E}$, such that the output of $\mathcal{E}(\eta)$ is distributed identically to E_η.
- **Theorem.** If $D^0 \approx D^1$ and E is polynomial-time constructible, then $F^0 \approx F^1$.

Proof via code modification

```
class SingleInd_{F^0,F^1} implements SingleEnv {
    private bitstring x, y;

    SingleInd_{F^0,F^1}(int η, bit b_0) {
        x ← D_η^{b_0};
        y ← E_η;
    }
    bitstring getX() {
        return (x, y);
    }
}
Call new SingleInd_{F^0,F^1}(η, 0)
```

Proof via code modification

```
class SingleInd_{F^0,F^1} implements SingleEnv {
    private bitstring x, y;

    SingleInd_{F^0,F^1}(int η, bit b_0) {
        x ← D_η^{b_0};
        y ← E_η;
    }
    bitstring getX() {
        return (x, y);
    }
}
Call new SingleInd_{F^0,F^1}(η, 0)
```

Propagate copies

Proof via code modification

```
class SingleInd_{F^0,F^1} implements SingleEnv {
    private bitstring x, y;

    SingleInd_{F^0,F^1}(int η, bit b_0) {
        x ← D^0_η;
        y ← E_η;
    }
    bitstring getX() {
        return (x, y);
    }
}
Call new SingleInd_{F^0,F^1}(η, 0)
```

Proof via code modification

```
class SingleInd_{F^0,F^1} implements SingleEnv {
    private bitstring x, y;

    SingleInd_{F^0,F^1}(int η, bit b_0) {
        x ← D^0_η;
        y ← E_η;
    }

    bitstring getX() {
        return (x, y);
    }
}
```

Call **new** SingleInd$_{F^0,F^1}(\eta, 0)$

Keep x inside SingleInd$_{D^0,D^1}(\eta, 0)$

Proof via code modification

class SingleInd$_{F^0,F^1}$ **implements** SingleEnv {
  private SingleEnv e;
  private bitstring y;

  SingleInd$_{F^0,F^1}$(**int** η, **bit** b_0) {
    $e :=$ **new** SingleInd$_{D^0,D^1}(\eta, 0)$;
    $y \leftarrow E_\eta$;
  }

  bitstring *getX*() {
    return $(e.$*getX*$(), y)$;
  }
}

Call **new** SingleInd$_{F^0,F^1}(\eta, 0)$

Proof via code modification

```
class SingleInd_{F^0,F^1} implements SingleEnv {
    private SingleEnv e;
    private bitstring y;

    SingleInd_{F^0,F^1}(int η, bit b_0) {
        e := new SingleInd_{D^0,D^1}(η, 0);
        y ← E_η;
    }
    bitstring getX() {
        return (e.getX(), y);
    }
}
```

Call **new** $\mathsf{SingleInd}_{F^0,F^1}(\eta, 0)$

Use $D^0 \approx D^1$

Proof via code modification

class SingleInd$_{F^0,F^1}$ **implements** SingleEnv {
  private SingleEnv e;
  private bitstring y;

  SingleInd$_{F^0,F^1}$(**int** η, **bit** b_0) {
    e := **new** SingleInd$_{D^0,D^1}(\eta, 1)$;
    $y \leftarrow E_\eta$;
  }

  bitstring *getX*() {
    return $(e.$*getX*$(), y)$;
  }
}

Call **new** SingleInd$_{F^0,F^1}(\eta, 0)$

Proof via code modification

```
class SingleInd_{F^0,F^1} implements SingleEnv {
    private SingleEnv e;
    private bitstring y;

    SingleInd_{F^0,F^1}(int η, bit b_0) {
        e := new SingleInd_{D^0,D^1}(η, 1);
        y ← E_η;
    }
    bitstring getX() {
        return (e.getX(), y);
    }
}
```

Call **new** $\mathsf{SingleInd}_{F^0,F^1}(\eta, 0)$

Take x out again

Proof via code modification

class SingleInd$_{F^0,F^1}$ **implements** SingleEnv {
    private bitstring x, y;

    SingleInd$_{F^0,F^1}$(**int** η, **bit** b_0) {
        $x \leftarrow D^1_\eta$;
        $y \leftarrow E_\eta$;
    }

    bitstring *getX*() {
        return (x, y);
    }
}

Call **new** SingleInd$_{F^0,F^1}(\eta, 0)$

This is equal to **new** SingleInd$_{F^0,F^1}(\eta, 1)$

“classical” proof

- Suppose that $F^0 \not\approx F^1$.
- Let $\mathcal{A}$ be a polynomial-time adversary such that $\mathcal{A}$ can distinguish D^0 and D^1 with non-negligible advantage.
  - $\mathcal{A}$ implements SimpleAdv

  Define the adversary $\mathcal{B}$ implementing SimpleAdv:

private SimpleAdv $\mathcal{A}$;

$\mathcal{B}$(SimpleAdv $\mathcal{A}_0$) {
  $\mathcal{A} := \mathcal{A}_0$;
}

bit *guess*(**int** η, SimpleEnv e) {
  ?????
}

- In *guess*, we could call $\mathcal{A}$.*guess*(e).
- But if e is SimpleInd$_{D^0,D^1}$ then the result probably won't make much sense.

Transforming the environment

```
class PairEnv implements SimpleEnv {
    private SimpleEnv e;;
    private bitstring y;;

    PairEnv(int η, SimpleEnv e₀) {
        e := e₀;
        y ← E_η;
    }

    bitstring getX() {
        return (e.getX(), y);
    }
}
```

The adversary $\mathcal{B}$

```
class 𝓑 implements SimpleAdv {
    private SimpleAdv 𝓐;

    𝓑(SimpleAdv 𝓐₀) {
        𝓐 := 𝓐₀;
    }

    bit guess(int η, SimpleEnv e) {
        return 𝓐.guess(new PairEnv(η, e));
    }
}
```

And now we have to argue that $\mathcal{B}$'s advantage really is the same as $\mathcal{A}$'s.

Multiple sampling

- Let $D^0 = \{D^0_\eta\}_{\eta\in\mathbb{N}}$ and $D^1 = \{D^1_\eta\}_{\eta\in\mathbb{N}}$ be two families of probability distributions.
- Let p be a positive polynomial.
- Let $\vec{D}^b_\eta$ be a probability distribution over tuples

$$(x_1, x_2, \ldots, x_{p(\eta)}) \in (\{0,1\}^*)^{p(\eta)}$$

  such that

  - each x_i is distributed according to D^b_η;
  - each x_i is is independent of all other x-s.

Multiple sampling

- Let $D^0 = \{D^0_\eta\}_{\eta\in\mathbb{N}}$ and $D^1 = \{D^1_\eta\}_{\eta\in\mathbb{N}}$ be two families of probability distributions.
- Let p be a positive polynomial.
- Let $\vec{D}^b_\eta$ be a probability distribution over tuples

$$(x_1, x_2, \ldots, x_{p(\eta)}) \in (\{0,1\}^*)^{p(\eta)}$$

  such that

  - each x_i is distributed according to D^b_η;
  - each x_i is is independent of all other x-s.

- To sample $\vec{D}^b_\eta$, sample D^b_η $p(\eta)$ times and construct the tuple of sampled values.

$\vec{D}$-s indistinguishable $\Rightarrow$ D-s indistinguishable

Theorem. If $\vec{D}^0 \approx \vec{D}^1$ then $D^0 \approx D^1$.

$\vec{D}$-s indistinguishable $\Rightarrow$ D-s indistinguishable

Theorem. If $\vec{D}^0 \approx \vec{D}^1$ then $D^0 \approx D^1$.
If ••• $\approx$ ••• then • $\approx$ •.

Contrapositive: if • $\not\approx$ • then ••• $\not\approx$ •••

$\vec{D}$-s indistinguishable $\Rightarrow$ D-s indistinguishable

Theorem. If $\vec{D}^0 \approx \vec{D}^1$ then $D^0 \approx D^1$.
If ●●● $\approx$ ●●● then ● $\approx$ ●.

Contrapositive: if ● $\not\approx$ ● then ●●● $\not\approx$ ●●●
If ● $\not\approx$ ● then there exists a PPT distinguisher $\mathcal{A}$:

$$\Pr[b = b^* \mid b \in_R \{0,1\}, x \leftarrow D_\eta^b, b^* \leftarrow \mathcal{A}(\eta, x)] \geq 1/2 + 1/q(\eta)$$

for some polynomial q and infinitely many η.

$\vec{D}$-s indistinguishable $\Rightarrow$ D-s indistinguishable

Theorem. If $\vec{D}^0 \approx \vec{D}^1$ then $D^0 \approx D^1$.
If ●●● ≈ ●●● then ● ≈ ●.

Contrapositive: if ● $\not\approx$ ● then ●●● $\not\approx$ ●●●
If ● $\not\approx$ ● then there exists a PPT distinguisher $\mathcal{A}$:

$$\Pr[\mathcal{A}(\eta, x) = 0 \,|\, x \leftarrow D^0_\eta] - \Pr[\mathcal{A}(\eta, x) = 0 \,|\, x \leftarrow D^1_\eta] \geq 2/q(\eta)$$

for some polynomial q and infinitely many η.

$\vec{D}$-s indistinguishable $\Rightarrow$ D-s indistinguishable

Theorem. If $\vec{D}^0 \approx \vec{D}^1$ then $D^0 \approx D^1$.
If ••• $\approx$ ••• then • $\approx$ •.

Contrapositive: if • $\not\approx$ • then ••• $\not\approx$ •••
If • $\not\approx$ • then there exists a PPT distinguisher $\mathcal{A}$:

$$\Pr[\mathcal{A}(\eta, x) = 0 \,|\, x \leftarrow D^0_\eta] - \Pr[\mathcal{A}(\eta, x) = 0 \,|\, x \leftarrow D^1_\eta] \geq 1/q(\eta)$$

for some polynomial q and infinitely many η.

$\vec{D}$-s indistinguishable $\Rightarrow$ D-s indistinguishable

Theorem. If $\vec{D}^0 \approx \vec{D}^1$ then $D^0 \approx D^1$.
If ●●● ≈ ●●● then ● ≈ ●.

Contrapositive: if ● $\not\approx$ ● then ●●● $\not\approx$ ●●●
If ● $\not\approx$ ● then there exists a PPT distinguisher $\mathcal{A}$:

$$\Pr[\mathcal{A}(\eta, x) = 0 \,|\, x \leftarrow D^0_\eta] - \Pr[\mathcal{A}(\eta, x) = 0 \,|\, x \leftarrow D^1_\eta] \geq 1/q(\eta)$$

for some polynomial q and infinitely many η.

Let $\mathcal{B}(\eta, (x_1, \ldots, x_{p(\eta)})) = \mathcal{A}(\eta, x_1)$.
Then $\mathcal{B}$ distinguishes ●●● and ●●●.

$\vec{D}$-s indistinguishable $\Rightarrow$ D-s indistinguishable

Theorem. If $\vec{D}^0 \approx \vec{D}^1$ then $D^0 \approx D^1$.
If ●●● $\approx$ ●●● then ● $\approx$ ●.

Contrapositive: if ● $\not\approx$ ● then ●●● $\not\approx$ ●●●
If ● $\not\approx$ ● then there exists a PPT distinguisher $\mathcal{A}$:

$$\Pr[\mathcal{A}(\eta, x) = 0 \,|\, x \leftarrow D_\eta^0] - \Pr[\mathcal{A}(\eta, x) = 0 \,|\, x \leftarrow D_\eta^1] \geq 1/q(\eta)$$

for some polynomial q and infinitely many η.

Let $\mathcal{B}(\eta, (x_1, \ldots, x_{p(\eta)})) = \mathcal{A}(\eta, x_1)$.
Then $\mathcal{B}$ distinguishes ●●● and ●●●.

I.e. we can distinguish ●●● from ●●● by just considering the first elements of the tuples.

D-s indistinguishable $\Rightarrow$ $\vec{D}$-s indistinguishable

(Interesting) theorem. If $D^0 \approx D^1$ and there exist polynomial-time algorithms $\mathcal{D}^0$ and $\mathcal{D}^1$, such that the output distribution of $\mathcal{D}^b(\eta)$ is equal to D^b_η, then $\vec{D}^0 \approx \vec{D}^1$.

D-s indistinguishable $\Rightarrow$ $\vec{D}$-s indistinguishable

(Interesting) theorem. If $D^0 \approx D^1$ and there exist polynomial-time algorithms $\mathcal{D}^0$ and $\mathcal{D}^1$, such that the output distribution of $\mathcal{D}^b(\eta)$ is equal to D^b_η, then $\vec{D}^0 \approx \vec{D}^1$.

Assume for now that the polynomial p is a constant. I.e. the length of the vector $\vec{x}$ does not depend on the security parameter η.
Let p be the common value of $p(\eta)$ for all η.

Theorem statement: if • $\approx$ • then ••• $\approx$ •••. (let $p = 3$)

D-s indistinguishable $\Rightarrow$ $\vec{D}$-s indistinguishable

(Interesting) theorem. If $D^0 \approx D^1$ and there exist polynomial-time algorithms $\mathcal{D}^0$ and $\mathcal{D}^1$, such that the output distribution of $\mathcal{D}^b(\eta)$ is equal to D^b_η, then $\vec{D}^0 \approx \vec{D}^1$.

Assume for now that the polynomial p is a constant. I.e. the length of the vector $\vec{x}$ does not depend on the security parameter η.
Let p be the common value of $p(\eta)$ for all η.

Theorem statement: if • ≈ • then ••• ≈ •••. (let $p = 3$)

Our lemmas said (• ≈ • ∧ • ≈ •) ⇒ • ≈ • and • ≈ • ⇒ •• ≈ ••.

D-s indistinguishable $\Rightarrow$ $\vec{D}$-s indistinguishable

(Interesting) theorem. If $D^0 \approx D^1$ and there exist polynomial-time algorithms $\mathcal{D}^0$ and $\mathcal{D}^1$, such that the output distribution of $\mathcal{D}^b(\eta)$ is equal to D^b_η, then $\vec{D}^0 \approx \vec{D}^1$.

Assume for now that the polynomial p is a constant. I.e. the length of the vector $\vec{x}$ does not depend on the security parameter η.
Let p be the common value of $p(\eta)$ for all η.

Theorem statement: if • $\approx$ • then ••• $\approx$ •••. (let $p = 3$)

Our lemmas said (• $\approx$ • $\wedge$ • $\approx$ •) $\Rightarrow$ • $\approx$ • and • $\approx$ • $\Rightarrow$ •• $\approx$ ••.

•••

D-s indistinguishable $\Rightarrow$ $\vec{D}$-s indistinguishable

(Interesting) theorem. If $D^0 \approx D^1$ and there exist polynomial-time algorithms $\mathcal{D}^0$ and $\mathcal{D}^1$, such that the output distribution of $\mathcal{D}^b(\eta)$ is equal to D^b_η, then $\vec{D}^0 \approx \vec{D}^1$.

Assume for now that the polynomial p is a constant. I.e. the length of the vector $\vec{x}$ does not depend on the security parameter η.
Let p be the common value of $p(\eta)$ for all η.

Theorem statement: if $\textcolor{red}{\bullet} \approx \textcolor{blue}{\bullet}$ then $\textcolor{red}{\bullet\bullet\bullet} \approx \textcolor{blue}{\bullet\bullet\bullet}$. (let $p = 3$)

Our lemmas said $(\textcolor{red}{\bullet} \approx \textcolor{green}{\bullet} \wedge \textcolor{green}{\bullet} \approx \textcolor{blue}{\bullet}) \Rightarrow \textcolor{red}{\bullet} \approx \textcolor{blue}{\bullet}$ and $\textcolor{red}{\bullet} \approx \textcolor{blue}{\bullet} \Rightarrow \textcolor{red}{\bullet}\textcolor{green}{\bullet} \approx \textcolor{blue}{\bullet}\textcolor{green}{\bullet}$.

$\textcolor{red}{\bullet\bullet\bullet} \approx \textcolor{red}{\bullet\bullet}\textcolor{blue}{\bullet}$

D-s indistinguishable $\Rightarrow$ $\vec{D}$-s indistinguishable

(Interesting) theorem. If $D^0 \approx D^1$ and there exist polynomial-time algorithms $\mathcal{D}^0$ and $\mathcal{D}^1$, such that the output distribution of $\mathcal{D}^b(\eta)$ is equal to D^b_η, then $\vec{D}^0 \approx \vec{D}^1$.

Assume for now that the polynomial p is a constant. I.e. the length of the vector $\vec{x}$ does not depend on the security parameter η.
Let p be the common value of $p(\eta)$ for all η.

Theorem statement: if ${\color{red}\bullet} \approx {\color{blue}\bullet}$ then ${\color{red}\bullet\bullet\bullet} \approx {\color{blue}\bullet\bullet\bullet}$. (let $p = 3$)

Our lemmas said $({\color{red}\bullet} \approx {\color{green}\bullet} \wedge {\color{green}\bullet} \approx {\color{blue}\bullet}) \Rightarrow {\color{red}\bullet} \approx {\color{blue}\bullet}$ and ${\color{red}\bullet} \approx {\color{blue}\bullet} \Rightarrow {\color{red}\bullet}{\color{green}\bullet} \approx {\color{blue}\bullet}{\color{green}\bullet}$.

${\color{red}\bullet\bullet\bullet} \approx {\color{red}\bullet\bullet}{\color{blue}\bullet} \approx {\color{red}\bullet}{\color{blue}\bullet\bullet}$

D-s indistinguishable $\Rightarrow$ $\vec{D}$-s indistinguishable

(Interesting) theorem. If $D^0 \approx D^1$ and there exist polynomial-time algorithms $\mathcal{D}^0$ and $\mathcal{D}^1$, such that the output distribution of $\mathcal{D}^b(\eta)$ is equal to D^b_η, then $\vec{D}^0 \approx \vec{D}^1$.

Assume for now that the polynomial p is a constant. I.e. the length of the vector $\vec{x}$ does not depend on the security parameter η.
Let p be the common value of $p(\eta)$ for all η.

Theorem statement: if $\textcolor{red}{\bullet} \approx \textcolor{blue}{\bullet}$ then $\textcolor{red}{\bullet\bullet\bullet} \approx \textcolor{blue}{\bullet\bullet\bullet}$. (let $p = 3$)

Our lemmas said $(\textcolor{red}{\bullet} \approx \textcolor{green}{\bullet} \wedge \textcolor{green}{\bullet} \approx \textcolor{blue}{\bullet}) \Rightarrow \textcolor{red}{\bullet} \approx \textcolor{blue}{\bullet}$ and $\textcolor{red}{\bullet} \approx \textcolor{blue}{\bullet} \Rightarrow \textcolor{red}{\bullet}\textcolor{green}{\bullet} \approx \textcolor{blue}{\bullet}\textcolor{green}{\bullet}$.

$\textcolor{red}{\bullet\bullet\bullet} \approx \textcolor{red}{\bullet\bullet}\textcolor{blue}{\bullet} \approx \textcolor{red}{\bullet}\textcolor{blue}{\bullet\bullet} \approx \textcolor{blue}{\bullet\bullet\bullet}$.

D-s indistinguishable $\Rightarrow$ $\vec{D}$-s indistinguishable

(Interesting) theorem. If $D^0 \approx D^1$ and there exist polynomial-time algorithms $\mathcal{D}^0$ and $\mathcal{D}^1$, such that the output distribution of $\mathcal{D}^b(\eta)$ is equal to D^b_η, then $\vec{D}^0 \approx \vec{D}^1$.

Assume for now that the polynomial p is a constant. I.e. the length of the vector $\vec{x}$ does not depend on the security parameter η.
Let p be the common value of $p(\eta)$ for all η.

Theorem statement: if $\textcolor{red}{\bullet} \approx \textcolor{blue}{\bullet}$ then $\textcolor{red}{\bullet\bullet\bullet} \approx \textcolor{blue}{\bullet\bullet\bullet}$. (let $p = 3$)

Our lemmas said $(\textcolor{red}{\bullet} \approx \textcolor{green}{\bullet} \wedge \textcolor{green}{\bullet} \approx \textcolor{blue}{\bullet}) \Rightarrow \textcolor{red}{\bullet} \approx \textcolor{blue}{\bullet}$ and $\textcolor{red}{\bullet} \approx \textcolor{blue}{\bullet} \Rightarrow \textcolor{red}{\bullet}\textcolor{green}{\bullet} \approx \textcolor{blue}{\bullet}\textcolor{green}{\bullet}$.

$\textcolor{red}{\bullet\bullet\bullet} \approx \textcolor{red}{\bullet\bullet}\textcolor{blue}{\bullet} \approx \textcolor{red}{\bullet}\textcolor{blue}{\bullet\bullet} \approx \textcolor{blue}{\bullet\bullet\bullet}$. By transitivity, $\textcolor{red}{\bullet\bullet\bullet} \approx \textcolor{blue}{\bullet\bullet\bullet}$.

(Actually, we're done with this case)

Constructing the distinguisher

Contrapositive: if $\textcolor{red}{\bullet\bullet\bullet} \not\approx \textcolor{blue}{\bullet\bullet\bullet}$ then $\textcolor{red}{\bullet} \not\approx \textcolor{blue}{\bullet}$.

Constructing the distinguisher

Contrapositive: if ●●● $\not\approx$ ●●● then ● $\not\approx$ ●.
If ●●● $\not\approx$ ●●● then there exists a PPT distinguisher $\mathcal{A}$:

$$\Pr[\mathcal{A}(\eta, \vec{x}) = 0 \mid \vec{x} \leftarrow \vec{D}^0_\eta] - \Pr[\mathcal{A}(\eta, \vec{x}) = 0 \mid \vec{x} \leftarrow \vec{D}^1_\eta] \geq 1/q(\eta)$$

for some polynomial q and infinitely many η.

Hybrid distributions

If ${\color{red}\bullet\bullet\bullet} \not\approx {\color{blue}\bullet\bullet\bullet}$ then

$$({\color{red}\bullet\bullet\bullet} \not\approx {\color{red}\bullet\bullet}{\color{blue}\bullet}) \vee ({\color{red}\bullet\bullet}{\color{blue}\bullet} \not\approx {\color{red}\bullet}{\color{blue}\bullet\bullet}) \vee ({\color{red}\bullet}{\color{blue}\bullet\bullet} \not\approx {\color{blue}\bullet\bullet\bullet})$$

Hybrid distributions

If ${\color{red}\bullet\bullet\bullet} \not\approx {\color{blue}\bullet\bullet\bullet}$ then

$$({\color{red}\bullet\bullet\bullet} \not\approx {\color{red}\bullet\bullet}{\color{blue}\bullet}) \vee ({\color{red}\bullet\bullet}{\color{blue}\bullet} \not\approx {\color{red}\bullet}{\color{blue}\bullet\bullet}) \vee ({\color{red}\bullet}{\color{blue}\bullet\bullet} \not\approx {\color{blue}\bullet\bullet\bullet})$$

Let $\vec{E}^k_\eta$, where $0 \leq k \leq p$, be a probability distribution over tuples $(x_1, \ldots, x_p)$, where

- each x_i is independent of all other x-s;
- $x_1, \ldots, x_k$ are distributed according to ${\color{red}D^0_\eta}$;
- $x_{k+1}, \ldots, x_p$ are distributed according to ${\color{blue}D^1_\eta}$.

Thus $\vec{E}^0_\eta = {\color{blue}\vec{D}^1_\eta}$ and $\vec{E}^p_\eta = {\color{red}\vec{D}^0_\eta}$. Define $P^k_\eta = \Pr[\mathcal{A}(\eta, \vec{x}) = 0 \,|\, \vec{x} \leftarrow \vec{E}^k_\eta]$. Then for infinitely many η:

$$1/q(\eta) \leq P^p_\eta - P^0_\eta = \sum_{i=1}^{p} (P^i_\eta - P^{i-1}_\eta) \ .$$

And for some j_η, $P^{j_\eta}_\eta - P^{j_\eta - 1}_\eta \geq 1/(p \cdot q(\eta))$.

$\mathcal{A}$ distinguishes hybrids

There exists j, such that $j = j_\eta$ for infinitely many η. Thus

$$\Pr[\mathcal{A}(\eta, \vec{x}) = 0 \,|\, \vec{x} \leftarrow \vec{E}^{j}_{\eta}] - \Pr[\mathcal{A}(\eta, \vec{x}) = 0 \,|\, \vec{x} \leftarrow \vec{E}^{j-1}_{\eta}] \geq 1/(p \cdot q(\eta))$$

for infinitely many η. We have $\vec{E}^{j-1} \not\approx \vec{E}^{j}$.

$\mathcal{A}$ distinguishes hybrids

There exists j, such that $j = j_\eta$ for infinitely many η. Thus

$$\Pr[\mathcal{A}(\eta, \vec{x}) = 0 \,|\, \vec{x} \leftarrow {\color{red}\vec{E}_\eta^j}] - \Pr[\mathcal{A}(\eta, \vec{x}) = 0 \,|\, \vec{x} \leftarrow {\color{blue}\vec{E}_\eta^{j-1}}] \geq 1/(p \cdot q(\eta))$$

for infinitely many η. We have $\vec{E}^{j-1} \not\approx \vec{E}^j$.

If we can distinguish

$$\vec{E}^j = \underbrace{{\color{red}\bullet\bullet} \cdots {\color{red}\bullet}}_{j-1} {\color{red}\bullet} \underbrace{{\color{blue}\bullet\bullet} \cdots {\color{blue}\bullet}}_{p-j}$$

from

$$\vec{E}^{j-1} = \underbrace{{\color{red}\bullet\bullet} \cdots {\color{red}\bullet}}_{j-1} {\color{blue}\bullet} \underbrace{{\color{blue}\bullet\bullet} \cdots {\color{blue}\bullet}}_{p-j}$$

using $\mathcal{A}$, then how do we distinguish ${\color{red}\bullet}$ and ${\color{blue}\bullet}$?

Distinguisher for D^0 and D^1

On input (η, x):

1. Let $x_1 := \mathcal{D}^0(\eta), \ldots, x_{j-1} := \mathcal{D}^0(\eta)$.
2. Let $x_j := x$
3. Let $x_{j+1} := \mathcal{D}^1(\eta), \ldots, x_p := \mathcal{D}^1(\eta)$
4. Let $\vec{x} = (x_1, \ldots, x_p)$.
5. Call $b^* := \mathcal{A}(\eta, \vec{x})$ and return b^*.

The advantage of this distinguisher is at least $1/(p \cdot q(\eta))$.

Distinguisher for D^0 and D^1

On input (η, x):

1. Let $x_1 := \mathcal{D}^0(\eta), \ldots, x_{j-1} := \mathcal{D}^0(\eta)$.
2. Let $x_j := x$
3. Let $x_{j+1} := \mathcal{D}^1(\eta), \ldots, x_p := \mathcal{D}^1(\eta)$
4. Let $\vec{x} = (x_1, \ldots, x_p)$.
5. Call $b^* := \mathcal{A}(\eta, \vec{x})$ and return b^*.

The advantage of this distinguisher is at least $1/(p \cdot q(\eta))$.

Unfortunately, the above construction was not constructive.

Being constructive

For infinitely many η we had

$$1/q(\eta) \leq P_\eta^p - P_\eta^0 = \sum_{i=1}^{p} (P_\eta^i - P_\eta^{i-1}) \quad .$$

Hence the <u>average</u> value of $P_\eta^j - P_\eta^{j-1}$ is $\geq 1/(p \cdot q(\eta))$.

Being constructive

For infinitely many η we had

$$1/q(\eta) \leq P_\eta^p - P_\eta^0 = \sum_{i=1}^{p} (P_\eta^i - P_\eta^{i-1}) \ .$$

Hence the average value of $P_\eta^j - P_\eta^{j-1}$ is $\geq 1/(p \cdot q(\eta))$.

Consider the following distinguisher $\mathcal{B}(\eta, x)$:

1. Let $j \in_R \{1, \ldots, p\}$.
2. Let $x_1 := \mathcal{D}^0(\eta), \ldots, x_{j-1} := \mathcal{D}^0(\eta)$.
3. Let $x_j := x$
4. Let $x_{j+1} := \mathcal{D}^1(\eta), \ldots, x_p := \mathcal{D}^1(\eta)$
5. Let $\vec{x} = (x_1, \ldots, x_p)$.
6. Call $b^* := \mathcal{A}(\eta, \vec{x})$ and return b^*.

What $\mathcal{B}$ does

If (for example) $p = 5$, then $\mathcal{B}$ tries to distinguish

$\color{blue}{\bullet\bullet\bullet\bullet\bullet}$ and $\color{red}{\bullet}\color{blue}{\bullet\bullet\bullet\bullet}$ with probability $1/5$
$\color{red}{\bullet}\color{blue}{\bullet\bullet\bullet\bullet}$ and $\color{red}{\bullet\bullet}\color{blue}{\bullet\bullet\bullet}$ with probability $1/5$
$\color{red}{\bullet\bullet}\color{blue}{\bullet\bullet\bullet}$ and $\color{red}{\bullet\bullet\bullet}\color{blue}{\bullet\bullet}$ with probability $1/5$
$\color{red}{\bullet\bullet\bullet}\color{blue}{\bullet\bullet}$ and $\color{red}{\bullet\bullet\bullet\bullet}\color{blue}{\bullet}$ with probability $1/5$
$\color{red}{\bullet\bullet\bullet\bullet}\color{blue}{\bullet}$ and $\color{red}{\bullet\bullet\bullet\bullet\bullet}$ with probability $1/5$

The advantage of $\mathcal{B}$ is $1/p$ times the sum of $\mathcal{A}$'s advantages of distinguishing these pairs of distributions.

The advantage of $\mathcal{B}$ is

$$\frac{1}{p}\sum_{j=1}^{p} P_\eta^j - P_\eta^{j-1} = \frac{1}{p}(P_\eta^p - P_\eta^0) \geq \frac{1}{p \cdot q(\eta)} \ .$$

If p depends on η

$\mathcal{B}(\eta, x)$ is:

1. Let $j \in_R \{1, \ldots, p(\eta)\}$.
2. Let $x_1 := \mathcal{D}^0(\eta), \ldots, x_{j-1} := \mathcal{D}^0(\eta)$.
3. Let $x_j := x$
4. Let $x_{j+1} := \mathcal{D}^1(\eta), \ldots, x_{p(\eta)} := \mathcal{D}^1(\eta)$
5. Let $\vec{x} = \big(x_1, \ldots, x_{p(\eta)}\big)$.
6. Call $b^* := \mathcal{A}(\eta, \vec{x})$ and return b^*.

The advantage of $\mathcal{B}$ is at least $1/(p(\eta) \cdot q(\eta))$.