BRAID-BASED CRYPTOLOGY

Patrick Dehornoy

http://www.math.unicaen.fr/∼dehornoy

Laboratoire de Mathématiques Nicolas Oresme, Caen

BRAID-BASED CRYPTOLOGY

Patrick Dehornoy

http://www.math.unicaen.fr/∼dehornoy

Laboratoire de Mathématiques Nicolas Oresme, Caen

- Introduction to braid groups;
- Description of some braid-based cryptographical protocols, after Sidelnokov & al. and Ko, Lee & al.;
- Length attack against the conjugacy problem, after Hofheinz–Steinwandt;
- A resisting protocol, after Sibert;
- New braid primitives: the shifted conjugacy problem;
- Discussion.

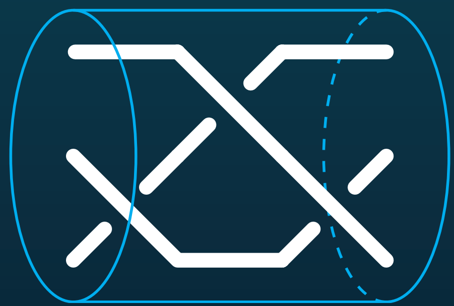
- A 4-strand **braid diagram** = 2D-projection of a 3D-figure

$\longleftarrow$

- A 4-strand **braid diagram** = 2D-projection of a 3D-figure

$\longleftarrow$

- isotopy = move the strands on the 3D-figure keeping the ends fixed

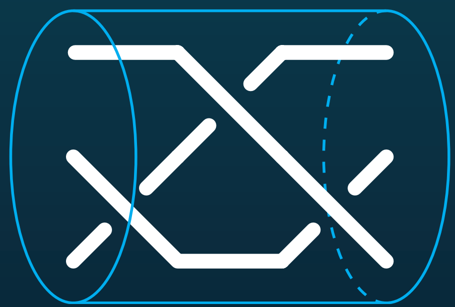


isotopic to

- A 4-strand **braid diagram** = 2D-projection of a 3D-figure

$\longleftarrow$

- isotopy = move the strands on the 3D-figure keeping the ends fixed

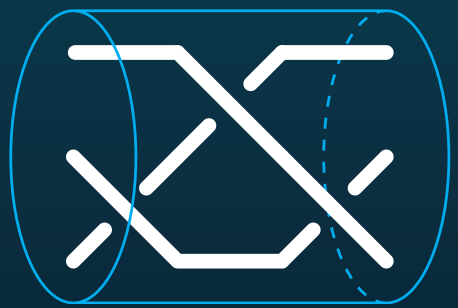


isotopic to

- A 4-strand **braid diagram** = 2D-projection of a 3D-figure

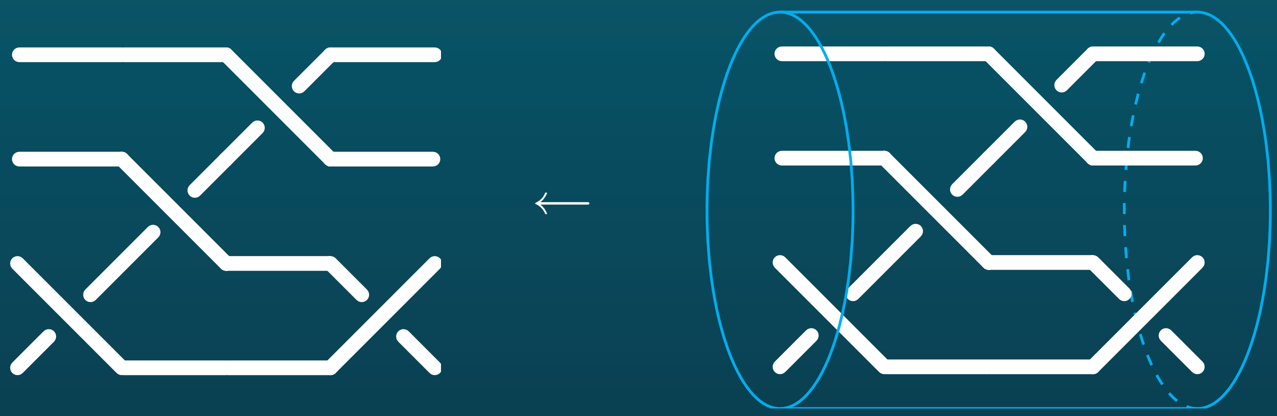
$\leftarrow$

- isotopy = move the strands on the 3D-figure keeping the ends fixed

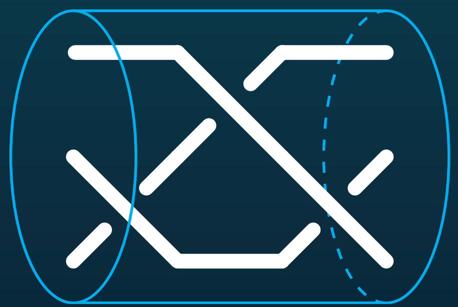


isotopic to

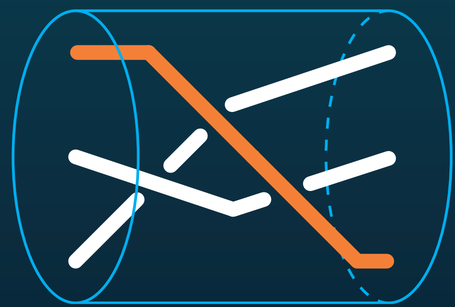
- A 4-strand **braid diagram** = 2D-projection of a 3D-figure



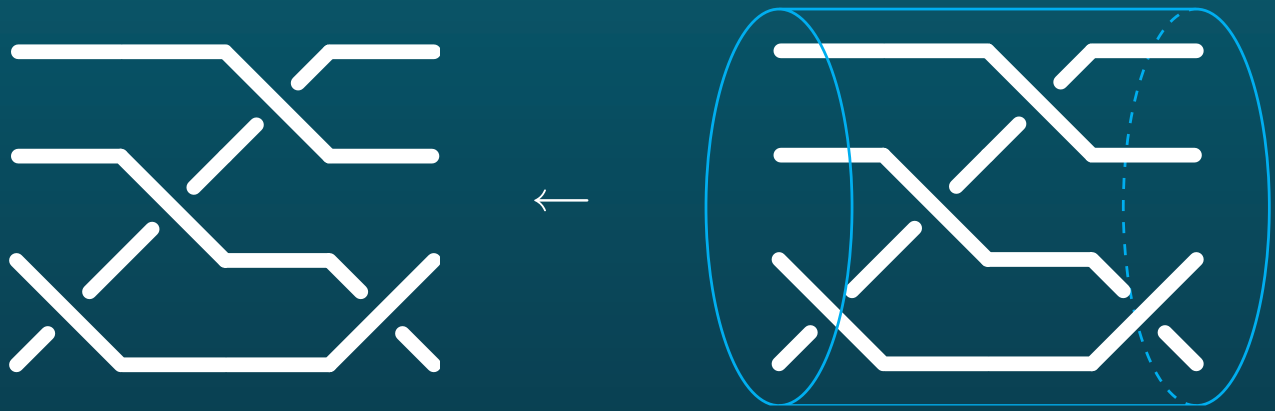
- isotopy = move the strands on the 3D-figure keeping the ends fixed



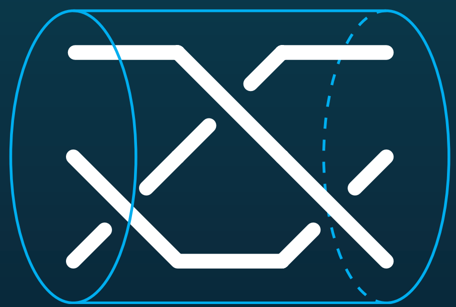
isotopic to



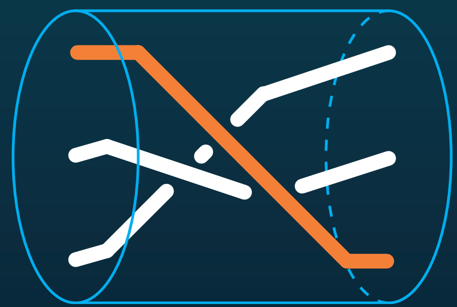
- A 4-strand **braid diagram** = 2D-projection of a 3D-figure



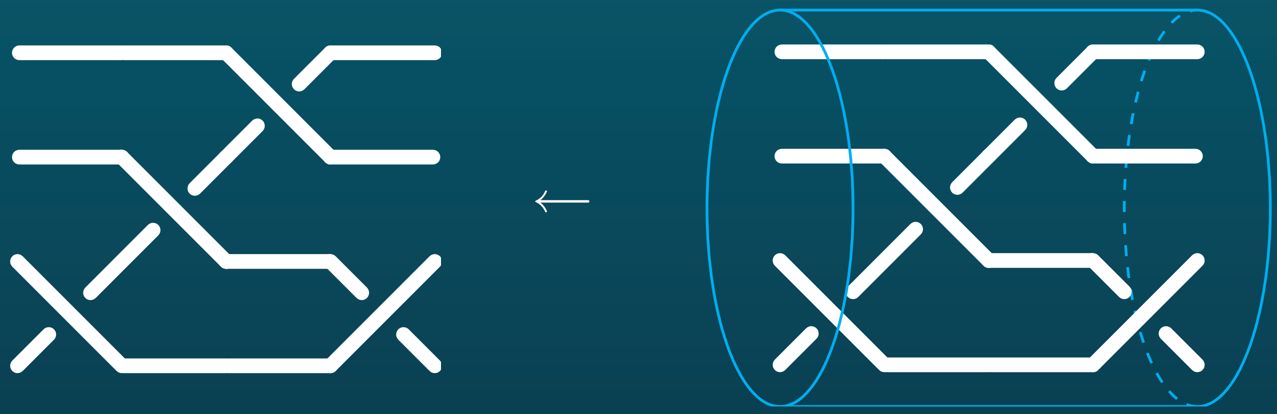
- isotopy = move the strands on the 3D-figure keeping the ends fixed



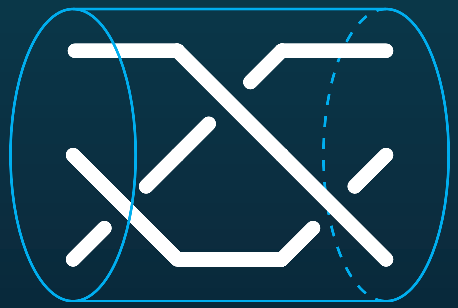
isotopic to



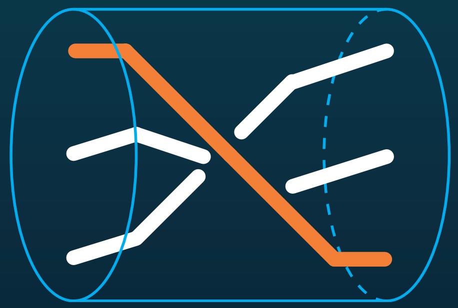
- A 4-strand **braid diagram** = 2D-projection of a 3D-figure



- isotopy = move the strands on the 3D-figure keeping the ends fixed



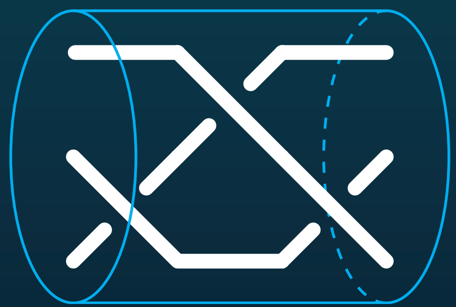
isotopic to



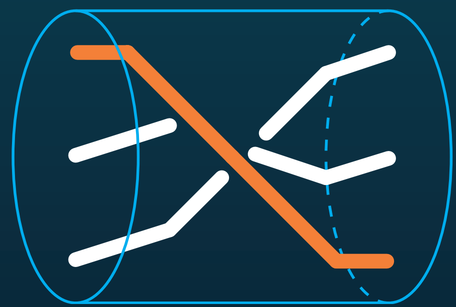
- A 4-strand **braid diagram** = 2D-projection of a 3D-figure

$\leftarrow$

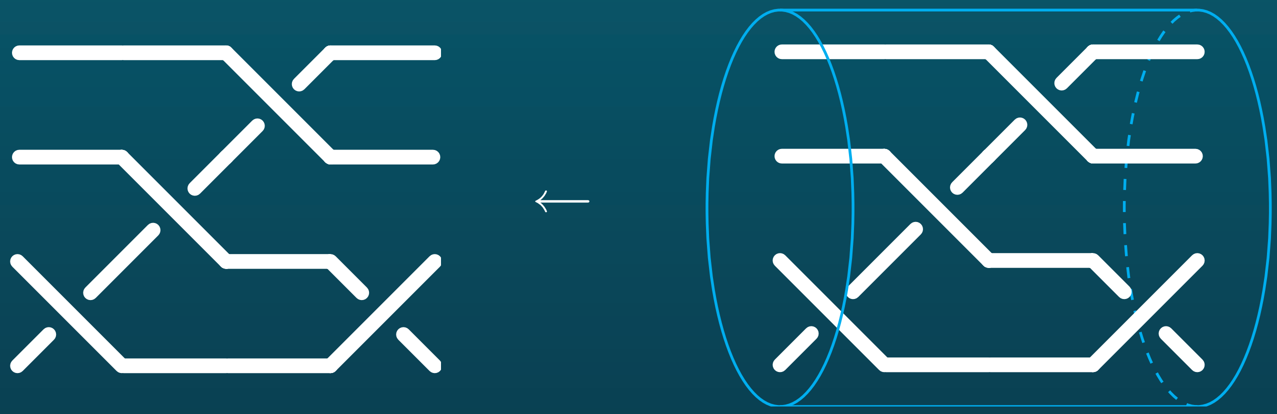
- isotopy = move the strands on the 3D-figure keeping the ends fixed



isotopic to

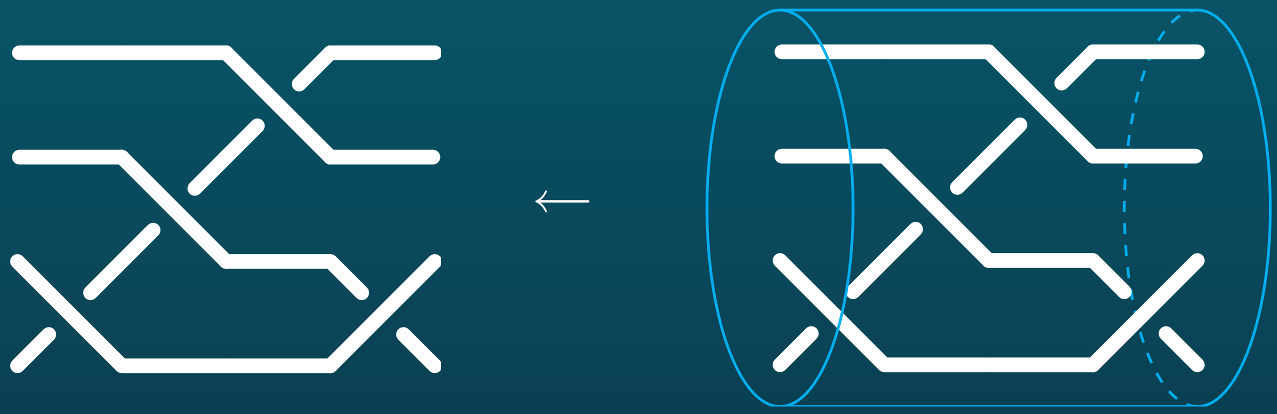


- A 4-strand **braid diagram** = 2D-projection of a 3D-figure



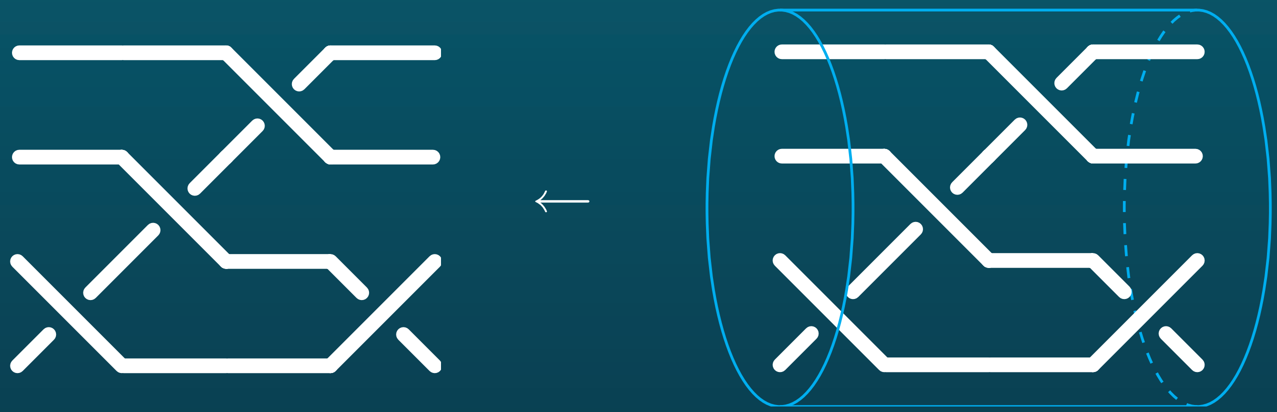
- isotopy = move the strands on the 3D-figure keeping the ends fixed

- A 4-strand **braid diagram** = 2D-projection of a 3D-figure



- isotopy = move the strands on the 3D-figure keeping the ends fixed

- **A 4-strand braid diagram = 2D-projection of a 3D-figure**

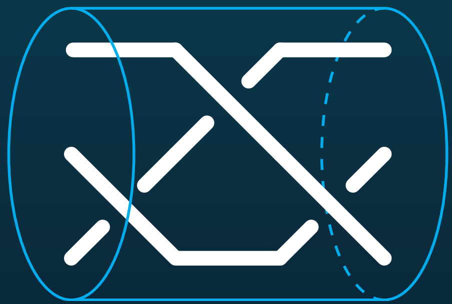


- **isotopy = move the strands on the 3D-figure keeping the ends fixed**

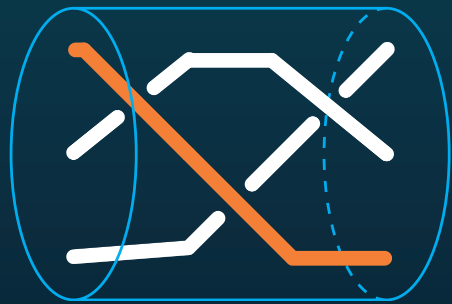
- A 4-strand **braid diagram** = 2D-projection of a 3D-figure

$\longleftarrow$

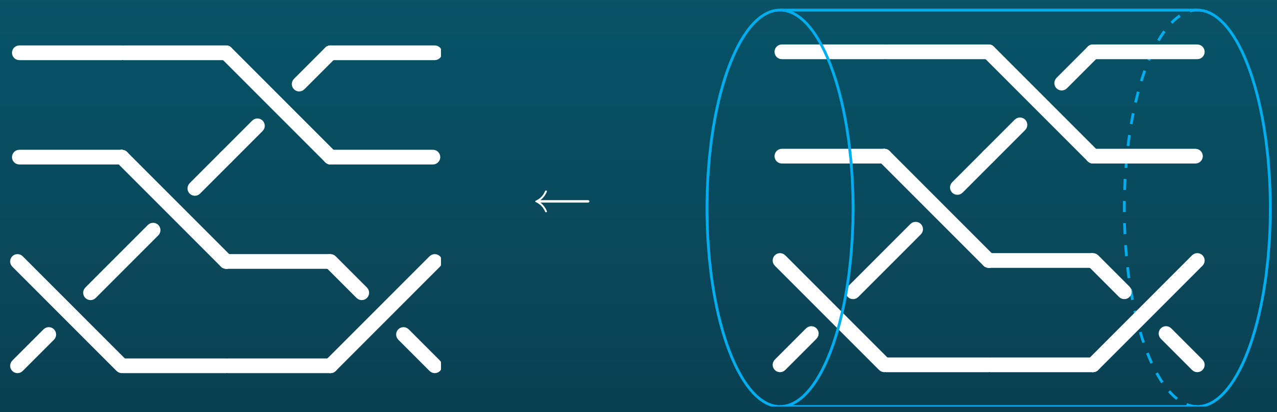
- isotopy = move the strands on the 3D-figure keeping the ends fixed



isotopic to

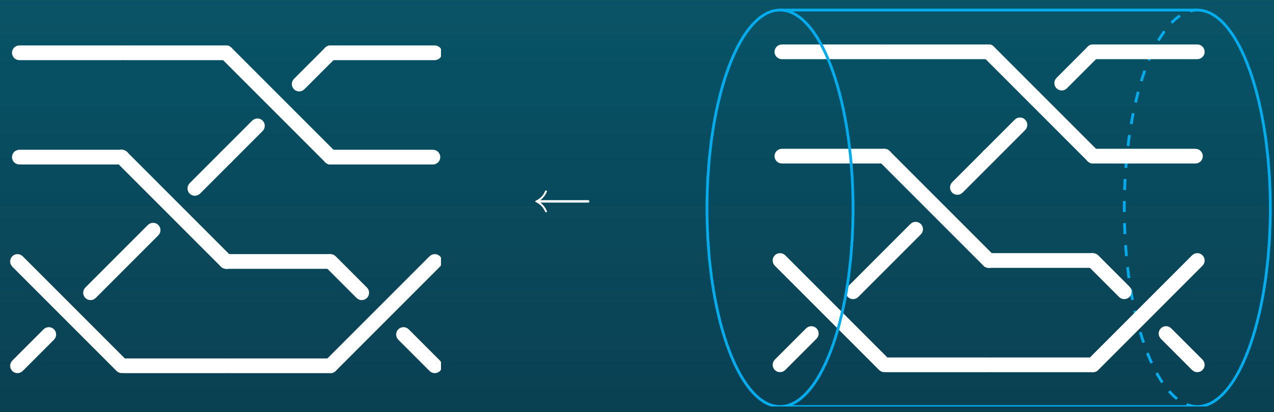


- A 4-strand **braid diagram** = 2D-projection of a 3D-figure

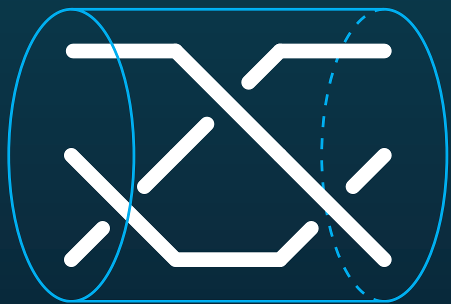


- isotopy = move the strands on the 3D-figure keeping the ends fixed

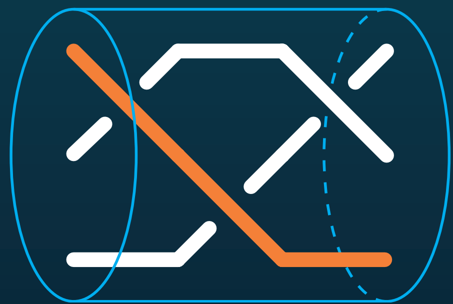
- A 4-strand **braid diagram** = 2D-projection of a 3D-figure



- isotopy = move the strands on the 3D-figure keeping the ends fixed



isotopic to

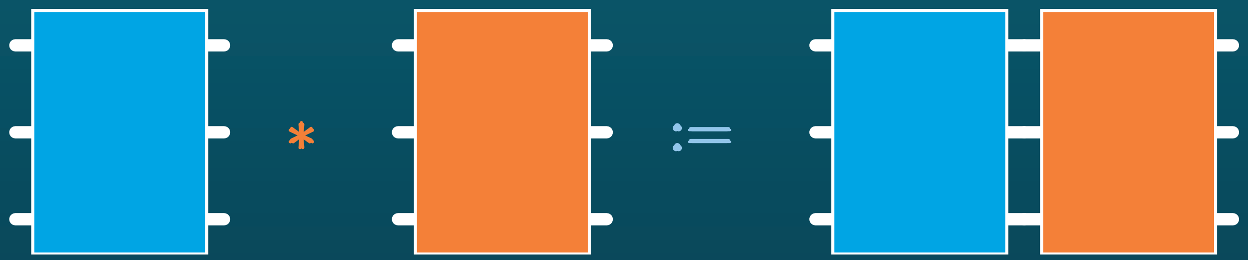


- a **braid** = an isotopy class

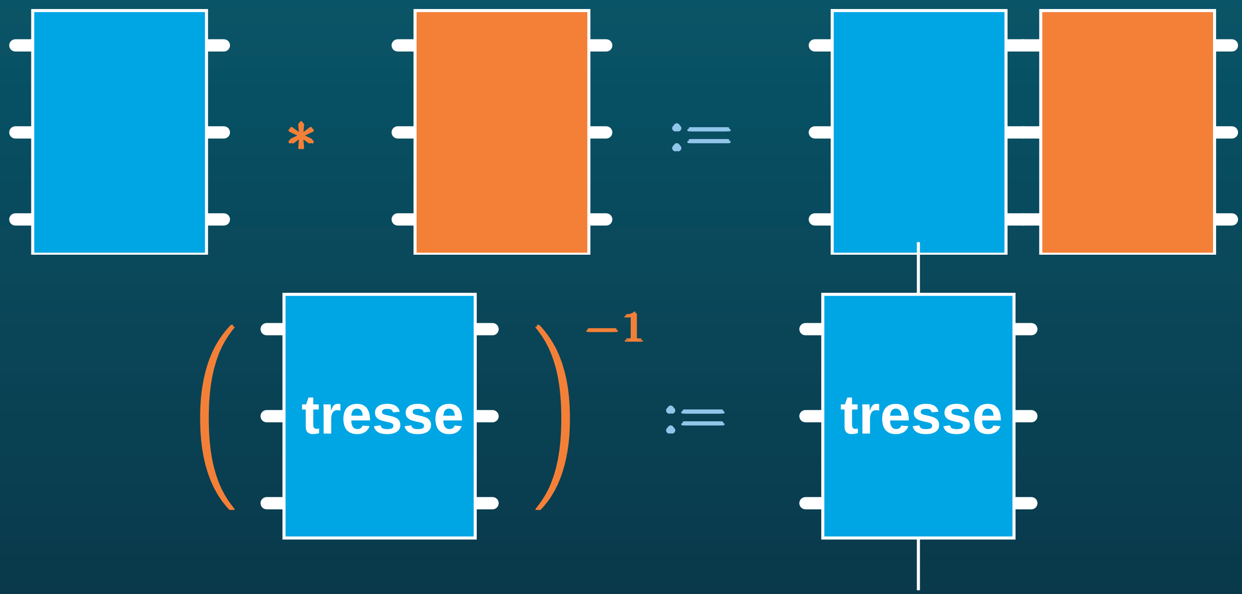
$\rightsquigarrow$ can be represented by 2D-diagram,

but different 2D-diagrams may give rise to the same braid.

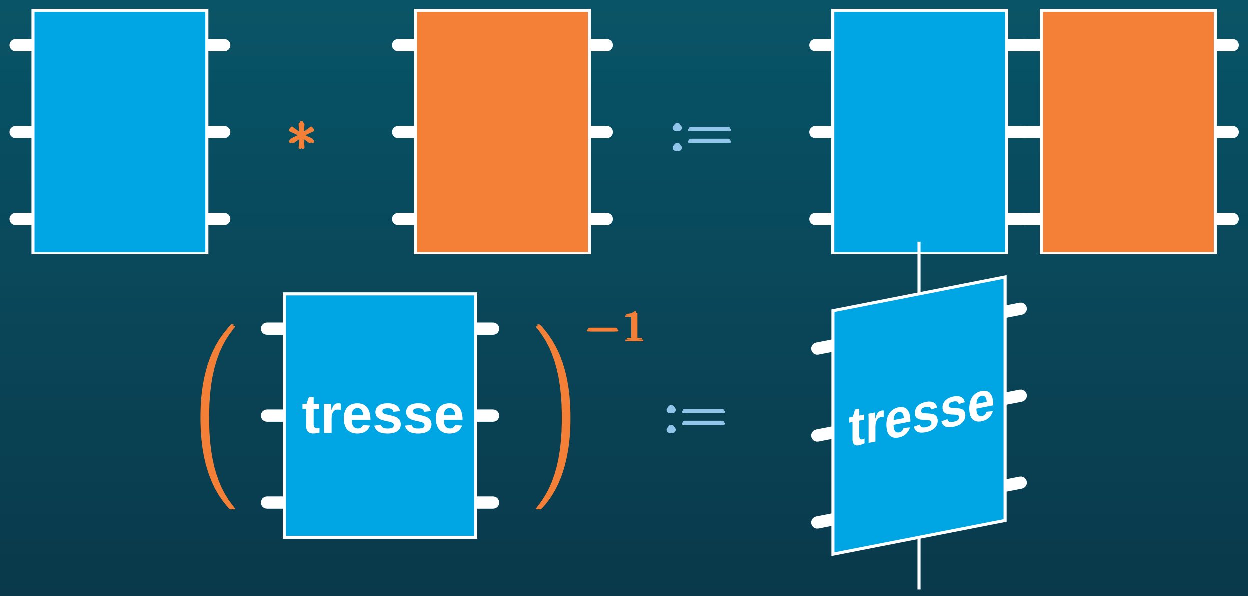
- **The product of two braids:**



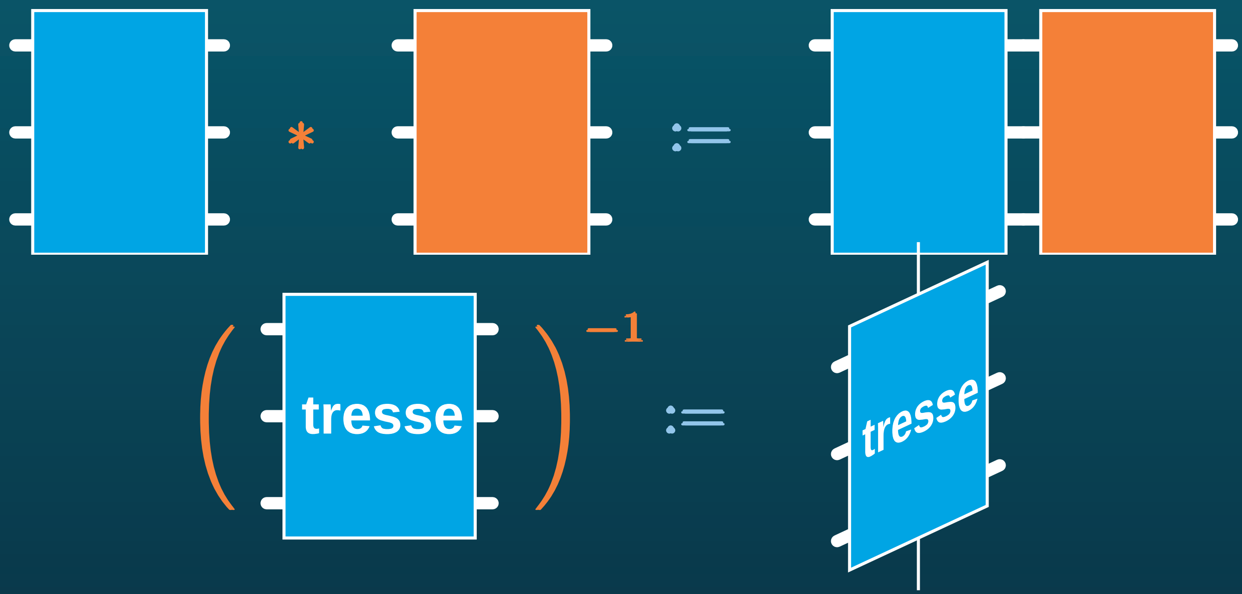
- **The product of two braids:**



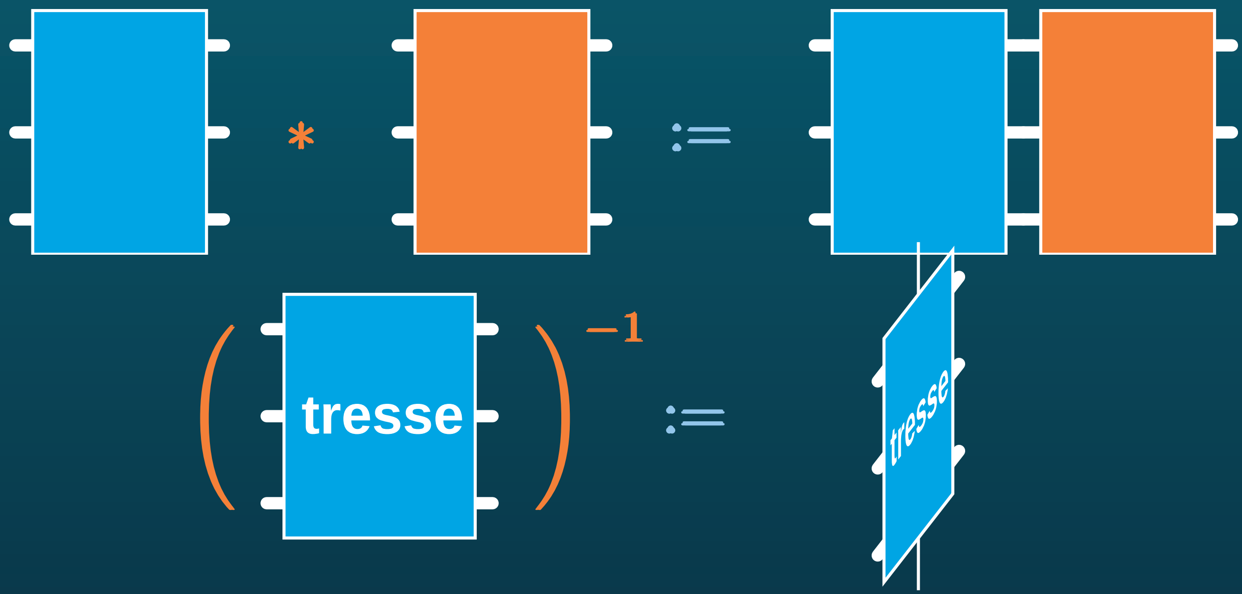
- **The product of two braids:**



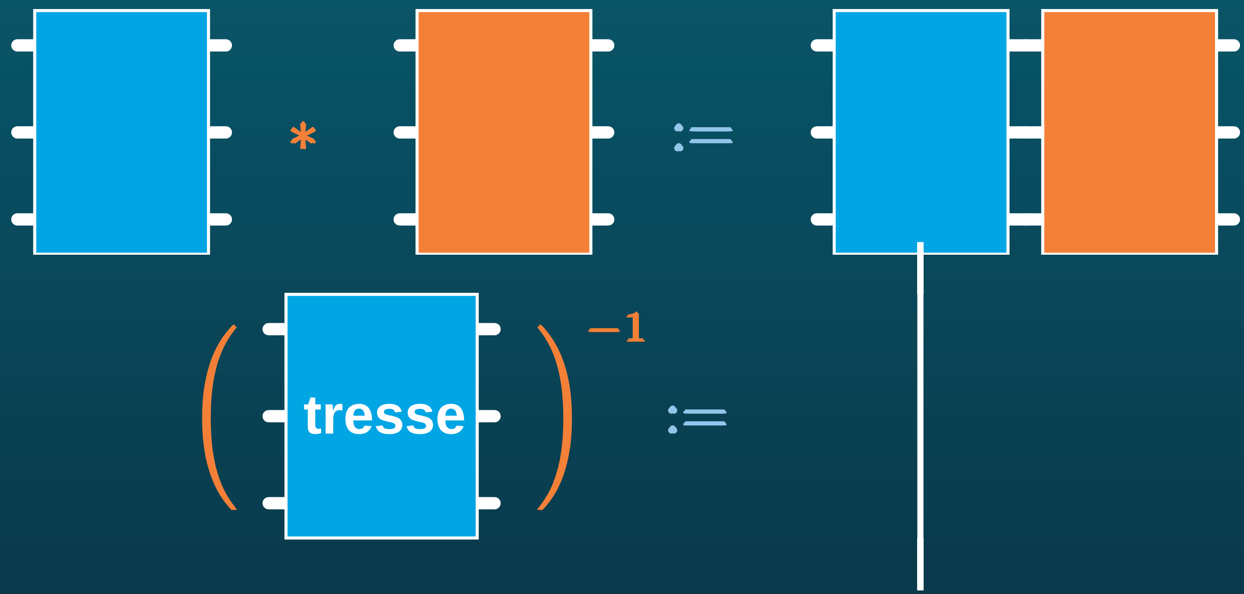
- **The product of two braids:**



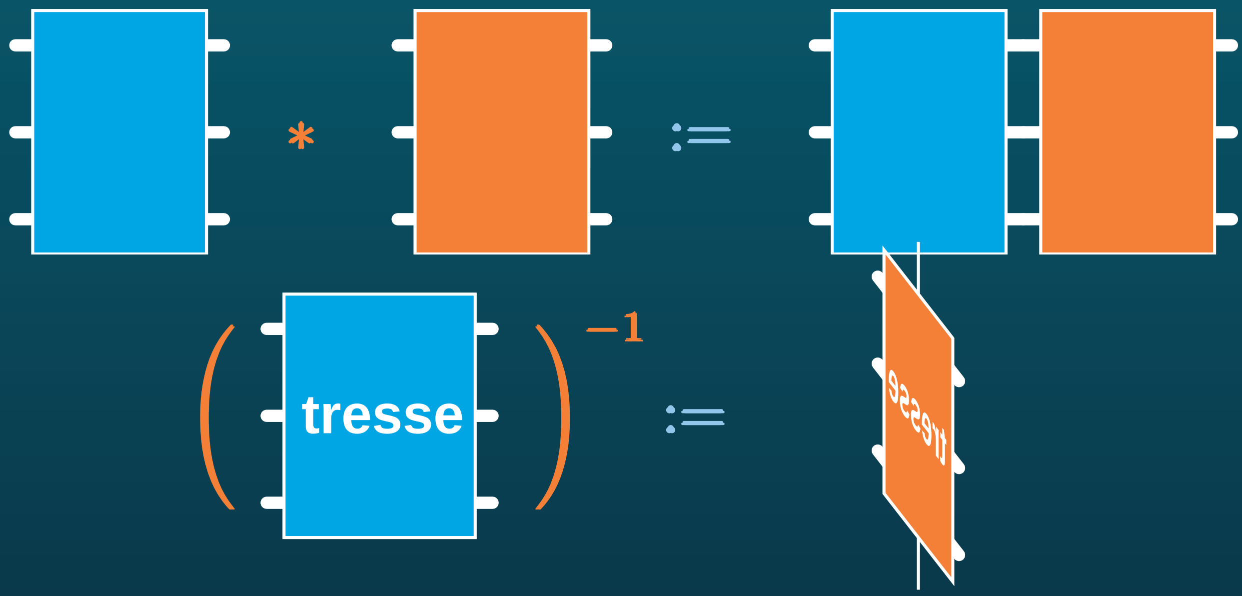
- **The product of two braids:**



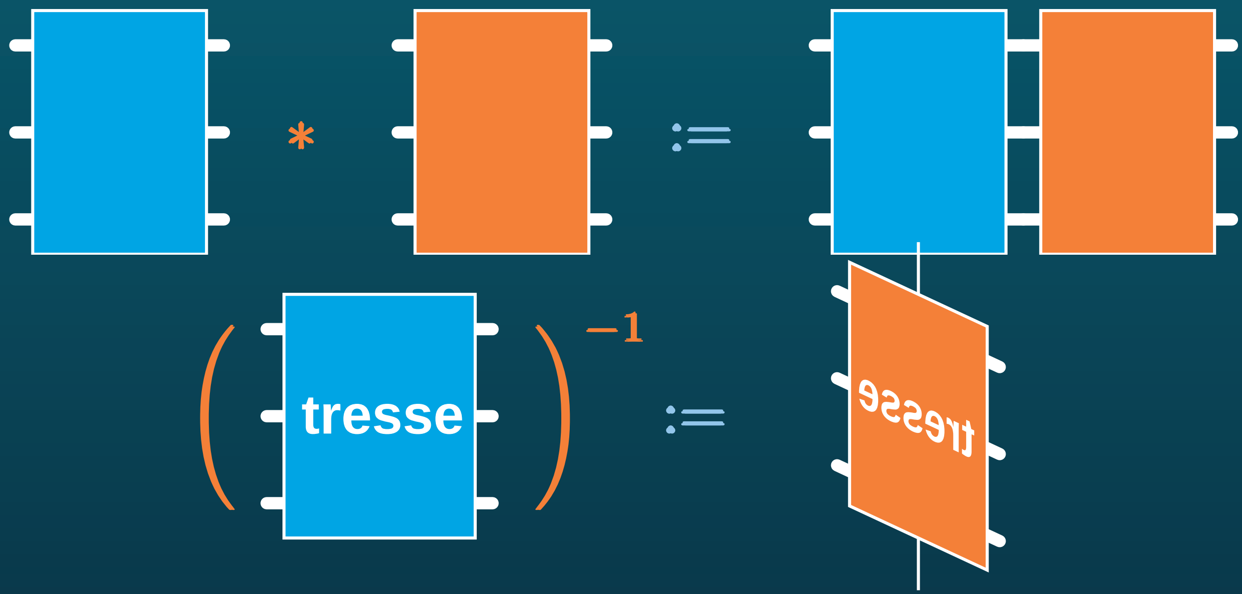
- **The product of two braids:**



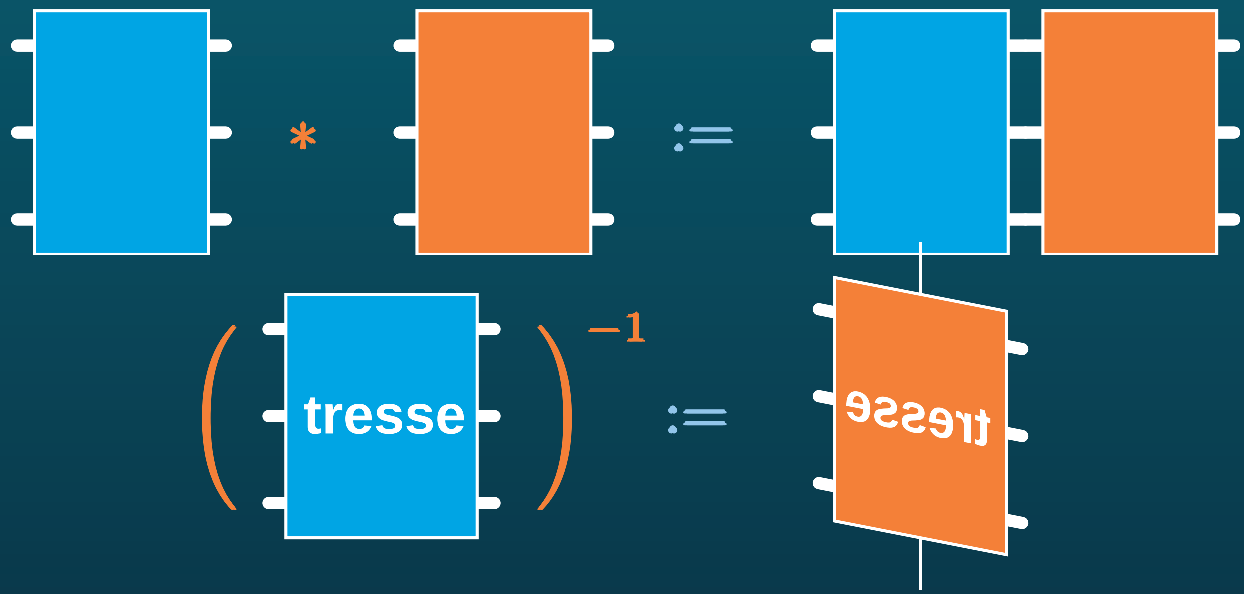
- **The product of two braids:**



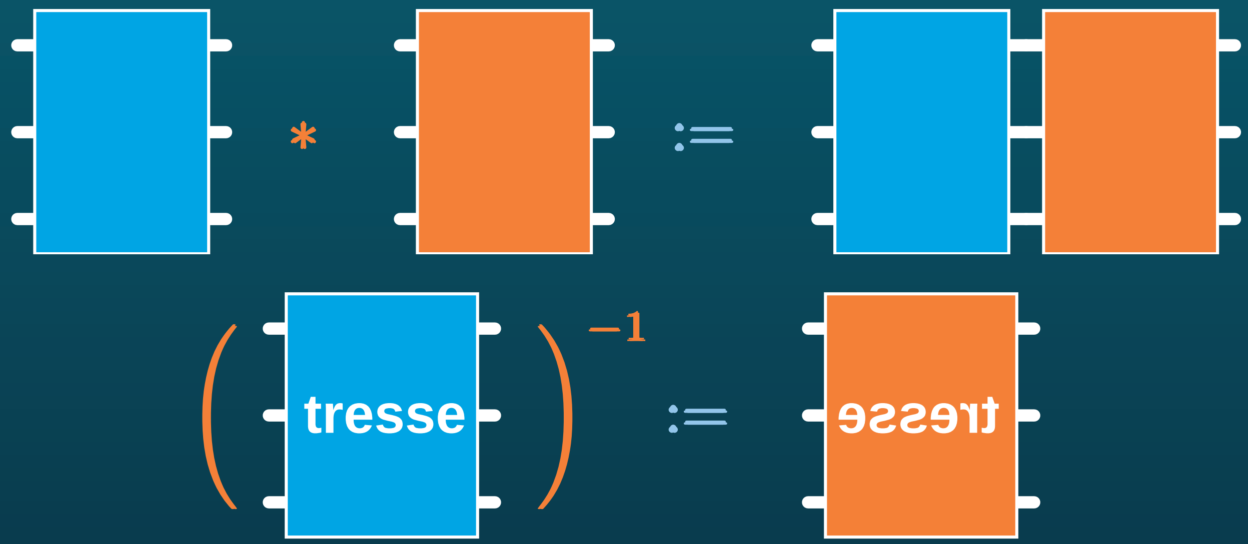
- **The product of two braids:**



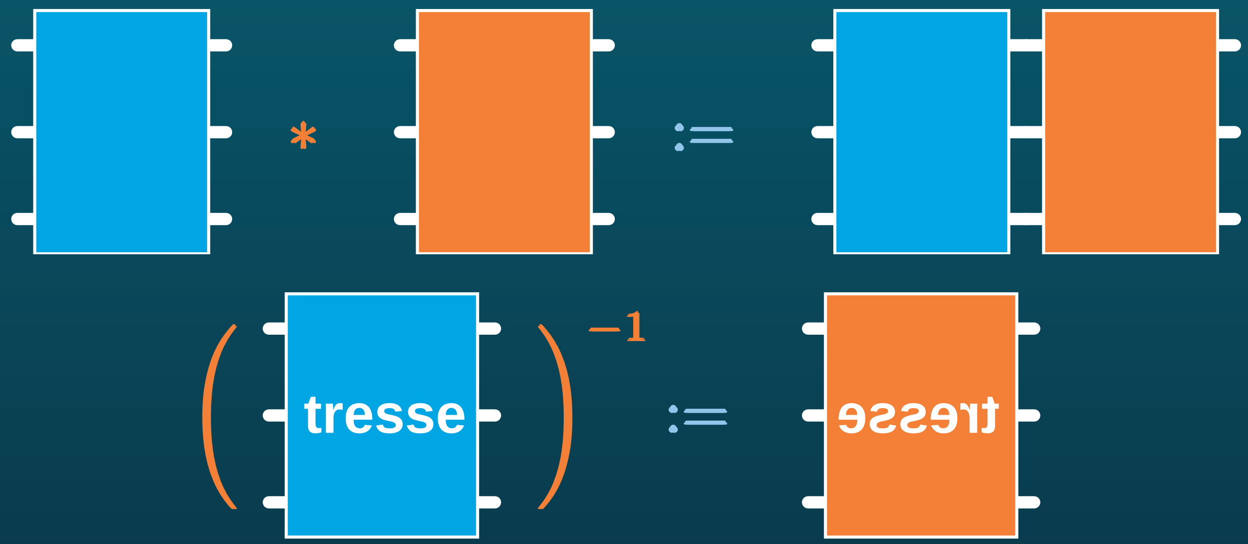
- **The product of two braids:**



- **The product of two braids:**

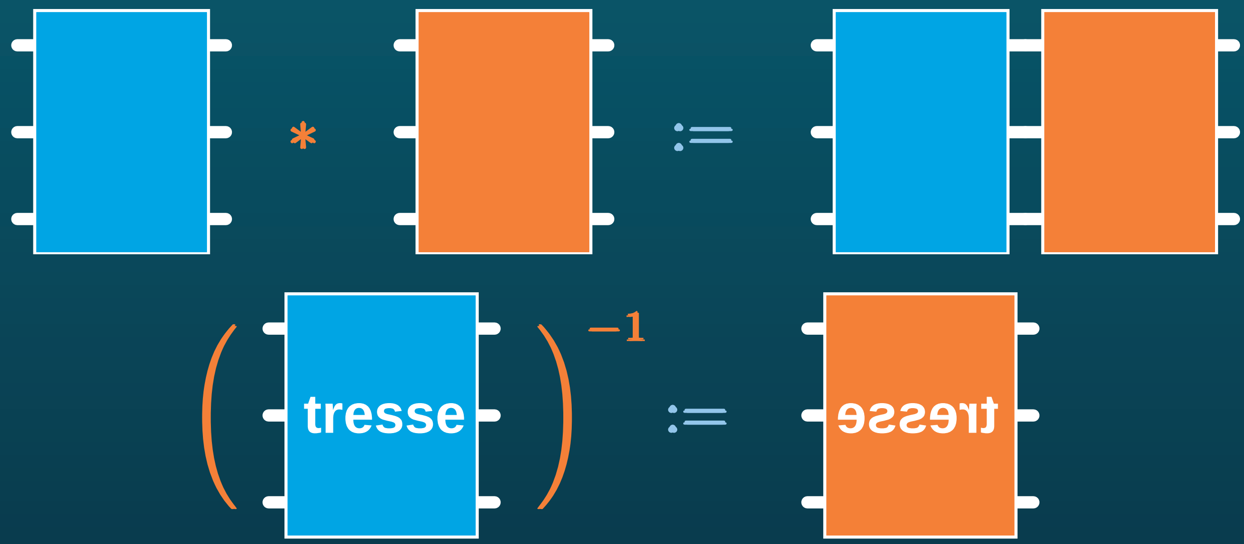


- **The product of two braids:**



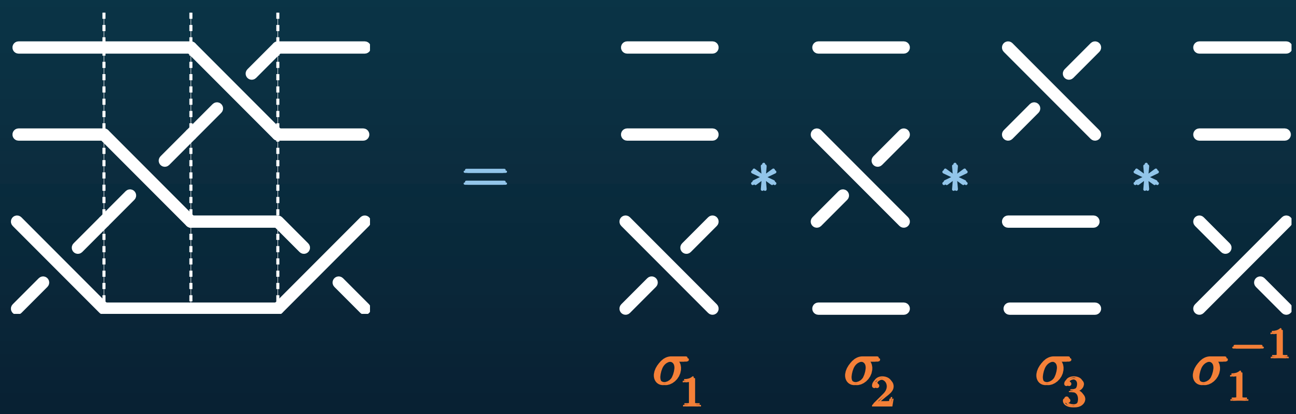
$\rightsquigarrow$ **For each n, a group: the group B_n of n strand braids (Emil Artin, $\sim$1925).**

- **The product of two braids:**

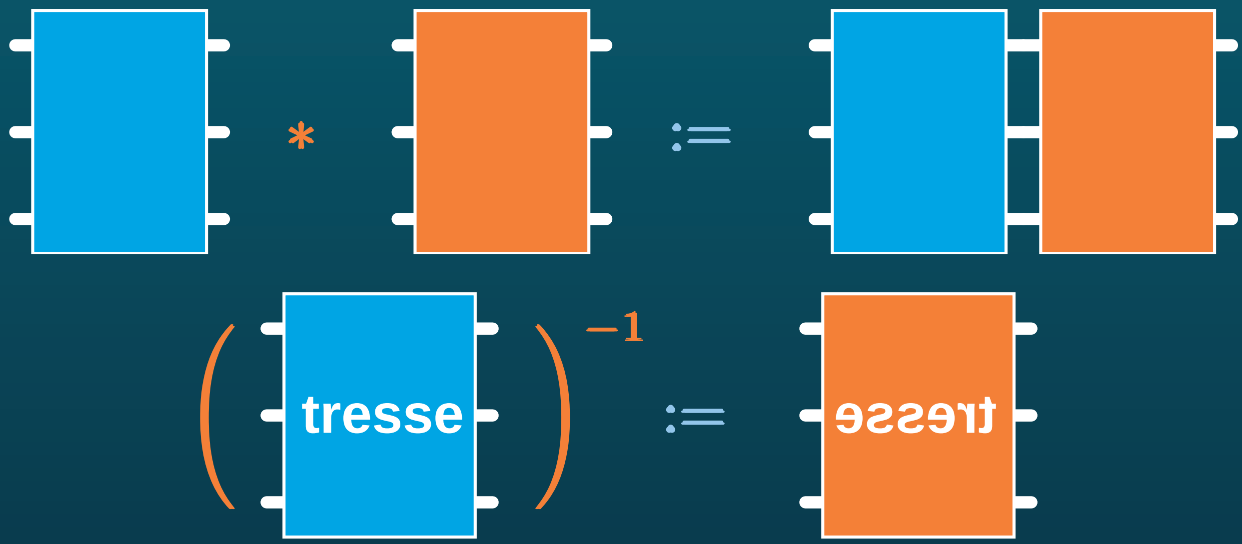


⇝ **For each n, a group: the group B_n of n strand braids (Emil Artin, $\sim$1925).**

- **Presentation of B_n:**

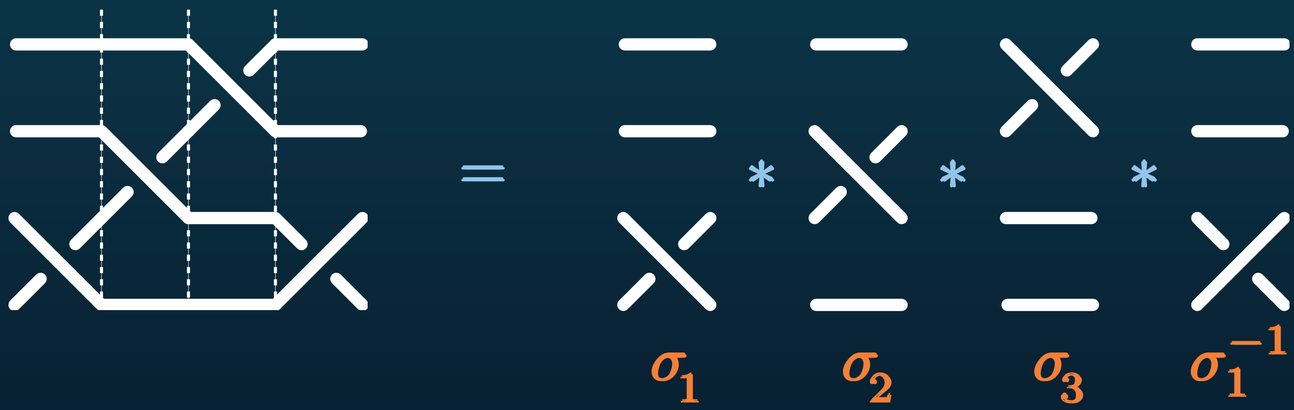


- **The product of two braids:**



$\rightsquigarrow$ **For each n, a group: the group B_n of n strand braids (Emil Artin, $\sim$1925).**

- **Presentation of B_n:**



- **Theorem (Artin): The braid group B_n is generated by $\sigma_1, \ldots, \sigma_{n-1}$, subject to the relations**

$$\sigma_i\sigma_j = \sigma_j\sigma_i \text{ with } |i-j| \geqslant 2, \quad \text{and} \quad \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j \text{ with } |i-j| = 1.$$

- **Problem: A and B wish to agree on a common secret, so that an intruder E cannot deduce the secret from the communication.**

- **Problem: A and B wish to agree on a common secret, so that an intruder E cannot deduce the secret from the communication.**

- **Notation:** LB_n **(**UB_n**) subgroup generated by** σ_1**, ...,** σ_{m-1} **(**σ_{m+1}**, ...,** σ_{n-1}**),** $m = \lfloor n/2 \rfloor$**.**

- **Problem:** A and B wish to agree on a common secret, so that an intruder E cannot deduce the secret from the communication.

- **Notation:** LB_n (UB_n) subgroup generated by σ_1, ..., σ_{m-1} (σ_{m+1}, ..., σ_{n-1}), $m = \lfloor n/2 \rfloor$.

- **Protocol** (Sidelnikov–Cherepnev–Yashchenko '93), (Ko–Lee–Cheon–Han–Kang–Park '00):
  - Key: p in B_n (public);

- **Problem:** A and B wish to agree on a common secret, so that an intruder E cannot deduce the secret from the communication.

- **Notation:** LB_n (UB_n) subgroup generated by σ_1, ..., σ_{m-1} (σ_{m+1}, ..., σ_{n-1}), $m = \lfloor n/2 \rfloor$.

- **Protocol** (Sidelnikov–Cherepnev–Yashchenko '93), (Ko–Lee–Cheon–Han–Kang–Park '00):
  - Key: p in B_n (public);
    - A chooses r in LB_n, and sends $p_A = rpr^{-1}$ to B;

- **Problem: A and B wish to agree on a common secret, so that an intruder E cannot deduce the secret from the communication.**

- **Notation:** LB_n **(**UB_n**) subgroup generated by** σ_1**, ...,** σ_{m-1} **(**σ_{m+1}**, ...,** σ_{n-1}**),** $m = \lfloor n/2 \rfloor$**.**

- **Protocol (Sidelnikov–Cherepnev–Yashchenko '93), (Ko–Lee–Cheon–Han–Kang–Park '00):**

- Key: p **in** B_n **(public);**

- A chooses r **in** LB_n**, and sends** $p_A = rpr^{-1}$ **to B;**

- B chooses s **in** UB_n**, and sends** $p_B = sps^{-1}$ **to A;**

- **Problem:** A and B wish to agree on a common secret, so that an intruder E cannot deduce the secret from the communication.

- **Notation:** LB_n (UB_n) subgroup generated by σ_1, ..., σ_{m-1} (σ_{m+1}, ..., σ_{n-1}), $m = \lfloor n/2 \rfloor$.

- **Protocol** (Sidelnikov–Cherepnev–Yashchenko '93), (Ko–Lee–Cheon–Han–Kang–Park '00):
  - Key: p in B_n (public);
    - A chooses r in LB_n, and sends $p_A = rpr^{-1}$ to B;
      - B chooses s in UB_n, and sends $p_B = sps^{-1}$ to A;
        - A computes $s_A = rp_Br^{-1}$;

- **Problem:** A and B wish to agree on a common secret, so that an intruder E cannot deduce the secret from the communication.

- **Notation:** LB_n (UB_n) subgroup generated by σ_1, ..., σ_{m-1} (σ_{m+1}, ..., σ_{n-1}), $m = \lfloor n/2 \rfloor$.

- **Protocol** (Sidelnikov–Cherepnev–Yashchenko ’93), (Ko–Lee–Cheon–Han–Kang–Park ’00):
  - Key: p in B_n (public);
    - A chooses r in LB_n, and sends $p_A = rpr^{-1}$ to B;
      - B chooses s in UB_n, and sends $p_B = sps^{-1}$ to A;
        - A computes $s_A = rp_Br^{-1}$;
          - B computes $s_B = sp_As^{-1}$.

- **Problem:** A and B wish to agree on a common secret, so that an intruder E cannot deduce the secret from the communication.

- **Notation:** LB_n (UB_n) subgroup generated by σ_1, ..., σ_{m-1} (σ_{m+1}, ..., σ_{n-1}), $m = \lfloor n/2 \rfloor$.

- **Protocol** (Sidelnikov–Cherepnev–Yashchenko '93), (Ko–Lee–Cheon–Han–Kang–Park '00):
  - Key: p in B_n (public);
    - A chooses r in LB_n, and sends $p_A = rpr^{-1}$ to B;
      - B chooses s in UB_n, and sends $p_B = sps^{-1}$ to A;
        - A computes $s_A = rp_Br^{-1}$;
          - B computes $s_B = sp_As^{-1}$.

- **Justification:** $rs = sr$, so $s_A = rsps^{-1}r^{-1} = srpr^{-1}s^{-1} = s_B$.

- **Problem:** A and B wish to agree on a common secret, so that an intruder E cannot deduce the secret from the communication.

- **Notation:** LB_n (UB_n) subgroup generated by σ_1, ..., σ_{m-1} (σ_{m+1}, ..., σ_{n-1}), $m = \lfloor n/2 \rfloor$.

- **Protocol** (Sidelnikov–Cherepnev–Yashchenko '93), (Ko–Lee–Cheon–Han–Kang–Park '00):
  - Key: p in B_n (public);
    - A chooses r in LB_n, and sends $p_A = rpr^{-1}$ to B;
      - B chooses s in UB_n, and sends $p_B = sps^{-1}$ to A;
        - A computes $s_A = rp_Br^{-1}$;
          - B computes $s_B = sp_As^{-1}$.

- **Justification:** $rs = sr$, so $s_A = rsps^{-1}r^{-1} = srpr^{-1}s^{-1} = s_B$.

- **Security:** Difficulty of retrieving x from (p, xpx^{-1}): the Conjugacy Search Problem.

- **Problem: A wishes to send a message m to B.**

  ↖ $\in \{0,1\}^*$

- **Problem:** **A wishes to send a message m to B.**
  $\nwarrow \in \{0,1\}^*$

- **Notation:** H **hash function** **from** B_n **to** $\{0,1\}^*$ **(= non-invertible + injective);**
  $\bigoplus$ **for "exclusive or".**

- **Problem: A wishes to send a message m to B.**

  $m \in \{0,1\}^*$

- **Notation: H hash function from B_n to $\{0,1\}^*$ (= non-invertible + injective);**
  $\oplus$ for "exclusive or".

- **Protocol (Ko–Lee & al. '00):**

- Keys: private: s in LB_n: only B knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;

- **Problem:** A wishes to send a message m to B. ($m \in \{0,1\}^*$)

- **Notation:** H hash function from B_n to $\{0,1\}^*$ (= non-invertible + injective);
  $\oplus$ for "exclusive or".

- **Protocol** (Ko–Lee & al. '00):

- Keys: private: s in LB_n: only B knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - A chooses r in UB_n, sends $p' = rpr^{-1}$ and $m' = m \oplus H(rqr^{-1})$;

- **Problem:** A wishes to send a message m to B.

  $m \in \{0,1\}^*$

- **Notation:** H hash function from B_n to $\{0,1\}^*$ (= non-invertible + injective);

  $\oplus$ for "exclusive or".

- **Protocol** (Ko–Lee & al. '00):

- Keys: private: s in LB_n: only B knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - A chooses r in UB_n, sends $p' = rpr^{-1}$ and $m' = m \oplus H(rqr^{-1})$;
    - B computes $m'' = m' \oplus H(sp's^{-1})$.

- **Problem: A wishes to send a message m to B.**
  ↖ $\in \{0,1\}^*$

- **Notation: H hash function from B_n to $\{0,1\}^*$ (= non-invertible + injective);**
  $\oplus$ for “exclusive or”.

- **Protocol (Ko–Lee & al. ’00):**

- Keys: private: s in LB_n: only B knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - A chooses r in UB_n, sends $p' = rpr^{-1}$ and $m' = m \oplus H(rqr^{-1})$;
    - B computes $m'' = m' \oplus H(sp's^{-1})$.

- **Justification: $rqr^{-1} = rsps^{-1}r^{-1} = srpr^{-1}s^{-1} = sp's^{-1}$, hence $m'' = m$.**

- **Problem:** A wishes to send a message m to B.

  ↖ $\in \{0,1\}^*$

- **Notation:** H hash function from B_n to $\{0,1\}^*$ (= non-invertible + injective);

  $\oplus$ for “exclusive or”.

- **Protocol** (Ko–Lee & al. ’00):

- Keys: private: s in LB_n: only B knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - A chooses r in UB_n, sends $p' = rpr^{-1}$ and $m' = m \oplus H(rqr^{-1})$;
    - B computes $m'' = m' \oplus H(sp's^{-1})$.

- **Justification:** $rqr^{-1} = rsps^{-1}r^{-1} = srpr^{-1}s^{-1} = sp's^{-1}$, hence $m'' = m$.

- **Security:** Difficulty of retrieving s from the pair (p, sps^{-1}): CSP again.

- **Problem:** The **prover** A wishes to prove her identity to the **verifier** B.

- **Problem:** The **prover** A wishes to prove her identity to the **verifier** B.

- **Protocol:**

- Keys: private: s in LB_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:**

- Keys: private: s in LB_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - B chooses r in UB_n, sends the challenge $x = rpr^{-1}$;

- **Problem: The prover A wishes to prove her identity to the verifier B.**

- **Protocol:**

- Keys: private: s in LB_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;

- B chooses r in UB_n, sends the challenge $x = rpr^{-1}$;

- A sends the response $y = sxs^{-1}$;

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:**

- Keys: private: s in LB_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - B chooses r in UB_n, sends the challenge $x = rpr^{-1}$;
    - A sends the response $y = sxs^{-1}$;
      - B checks $y = rqr^{-1}$.

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:**

- Keys: private: s in LB_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - B chooses r in UB_n, sends the challenge $x = rpr^{-1}$;
    - A sends the response $y = sxs^{-1}$;
      - B checks $y = rqr^{-1}$.

- **Justification:** $y = rqr^{-1} = rsps^{-1}r^{-1} = srpr^{-1}s^{-1} = sxs^{-1}$.

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:**

- Keys: private: s in LB_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - B chooses r in UB_n, sends the challenge $x = rpr^{-1}$;
    - A sends the response $y = sxs^{-1}$;
      - B checks $y = rqr^{-1}$.

- **Justification:** $y = rqr^{-1} = rsps^{-1}r^{-1} = srpr^{-1}s^{-1} = sxs^{-1}$.

- **Improvement:** A sends $H(sxs^{-1})$, and B checks $y = H(rqr^{-1})$ with H a hash function.

AUTHENTICATION (bis)

- **Problem: The prover A wishes to prove her identity to the verifier B.**

- **Problem:** The prover A wishes to prove her identity to the verifier B.
- **Protocol:** (Sibert-D.-Girault ’02, after Fiat-Shamir)

- **Problem: The prover A wishes to prove her identity to the verifier B.**

- **Protocol: (Sibert-D.-Girault '02, after Fiat-Shamir)**

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;

- **Problem: The prover A wishes to prove her identity to the verifier B.**

- **Protocol: (Sibert-D.-Girault '02, after Fiat-Shamir)**

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;

- Repeat k times the sequence:

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:** (Sibert-D.-Girault '02, after Fiat-Shamir)

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitment $x = rpr^{-1}$;

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:** (Sibert-D.-Girault '02, after Fiat-Shamir)

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
    - Repeat k times the sequence:
        - (i) A chooses r in B_n, and sends the commitment $x = rpr^{-1}$;
        - (ii) B chooses c in $\{0, 1\}$, and sends c;

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:** (Sibert-D.-Girault '02, after Fiat-Shamir)

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
    - Repeat k times the sequence:
        - (i) A chooses r in B_n, and sends the commitment $x = rpr^{-1}$;
        - (ii) B chooses c in $\{0, 1\}$, and sends c;
        - (iii) case $c = 0$
            - A sends $y = r$;
            - B checks $x = ypy^{-1}$;

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:** (Sibert-D.-Girault '02, after Fiat-Shamir)

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitment $x = rpr^{-1}$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;
    - (iii) case $c = 0$
      - A sends $y = r$;
      - B checks $x = ypy^{-1}$;

      case $c = 1$
      - A sends $y = rs^{-1}$;
      - B checks $x = yqy^{-1}$.

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:** (Sibert-D.-Girault '02, after Fiat-Shamir)

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
    - Repeat k times the sequence:
        - (i) A chooses r in B_n, and sends the commitment $x = rpr^{-1}$;
        - (ii) B chooses c in $\{0, 1\}$, and sends c;
        - (iii) case $c = 0$
            - A sends $y = r$;
            - B checks $x = ypy^{-1}$;
        - case $c = 1$
            - A sends $y = rs^{-1}$;
            - B checks $x = yqy^{-1}$.

- **Justification (case $c = 1$):** $x = rpr^{-1} = (rs^{-1})(sps^{-1})(sr^{-1}) = yqy^{-1}$,

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:** (Sibert-D.-Girault ’02, after Fiat-Shamir)

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitment $x = rpr^{-1}$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;
    - (iii) case $c = 0$
      - A sends $y = r$;
      - B checks $x = ypy^{-1}$;

      case $c = 1$
      - A sends $y = rs^{-1}$;
      - B checks $x = yqy^{-1}$.

- **Justification** (case $c = 1$): $x = rpr^{-1} = (rs^{-1})(sps^{-1})(sr^{-1}) = yqy^{-1}$,

  $\rightsquigarrow$ probability that A succeeds without knowing s is $\leqslant 1/2^k$.

- **Problem:** The prover A wishes to prove her identity to the verifier B.

- **Protocol:** (Sibert-D.-Girault '02, after Fiat-Shamir)

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = sps^{-1}$;
    - Repeat k times the sequence:
        - (i) A chooses r in B_n, and sends the commitment $x = rpr^{-1}$;
        - (ii) B chooses c in $\{0, 1\}$, and sends c;
        - (iii) case $c = 0$
            - A sends $y = r$;
            - B checks $x = ypy^{-1}$;

          case $c = 1$
            - A sends $y = rs^{-1}$;
            - B checks $x = yqy^{-1}$.

- **Justification** (case $c = 1$): $x = rpr^{-1} = (rs^{-1})(sps^{-1})(sr^{-1}) = yqy^{-1}$,

  $\rightsquigarrow$ probability that A succeeds without knowing s is $\leqslant 1/2^k$.

- **Improvement:** Replace x with $H(x)$.

- Security of the previous protocols: all relie on the difficulty of Conjugacy Search Problem:
  Assuming that p and q are conjugate in B_n, find s satisfying $q = sps^{-1}$.

- **Security of the previous protocols: all relie on the difficulty of Conjugacy Search Problem:**

  Assuming that p and q are conjugate in B_n, find s satisfying $q = sps^{-1}$.

  $\rightsquigarrow$ **more generally: the conjugacy problem of B_n.**

- **Security of the previous protocols: all relie on the difficulty of Conjugacy Search Problem:**
  Assuming that p and q are conjugate in B_n, find s satisfying $q = sps^{-1}$.
  $\rightsquigarrow$ more generally: the conjugacy problem of B_n.

- **Theorem (Garside, 1969): The conjugacy problem of B_n is solvable.**

- **Security of the previous protocols: all relie on the difficulty of Conjugacy Search Problem:**

  Assuming that p and q are conjugate in B_n, find s satisfying $q = sps^{-1}$.

  ⇝ more generally: the conjugacy problem of B_n.

- **Theorem (Garside, 1969): The conjugacy problem of B_n is solvable.**

⇝ Proposition: For each braid b, there exists a finite, effectively computable subset $SS(b)$ of the conjugacy class of b — "summit set" of b — s.t. b, b' are conjugate iff $SS(b') = SS(b)$.

- **Security of the previous protocols: all relie on the difficulty of Conjugacy Search Problem:**

  Assuming that p and q are conjugate in B_n, find s satisfying $q = sps^{-1}$.

  ⇝ more generally: the conjugacy problem of B_n.

- **Theorem (Garside, 1969): The conjugacy problem of B_n is solvable.**

⇝ Proposition: For each braid b, there exists a finite, effectively computable subset $SS(b)$ of the conjugacy class of b — "summit set" of b — s.t. b, b' are conjugate iff $SS(b') = SS(b)$.

- **In practice: $SS(b)$ is very large (exponential in the size of b),**

- **Security of the previous protocols: all relie on the difficulty of Conjugacy Search Problem:**

  Assuming that p and q are conjugate in B_n, find s satisfying $q = sps^{-1}$.

  $\rightsquigarrow$ more generally: the conjugacy problem of B_n.

- **Theorem (Garside, 1969): The conjugacy problem of B_n is solvable.**

$\rightsquigarrow$ Proposition: For each braid b, there exists a finite, effectively computable subset $SS(b)$ of the conjugacy class of b — "summit set" of b — s.t. b, b' are conjugate iff $SS(b') = SS(b)$.

- **In practice: $SS(b)$ is very large (exponential in the size of b),**

  but improvements: ElRifai–Morton, Gonzalez-Meneses, Gebhardt,...

  $\rightsquigarrow$ replace $SS(b)$ with smaller subsets $SSS(b)$, then $USS(b)$...

  that can be computed more easily

- **Security of the previous protocols:** all relie on the difficulty of Conjugacy Search Problem:

  Assuming that p and q are conjugate in B_n, find s satisfying $q = sps^{-1}$.

  ⇝ more generally: the conjugacy problem of B_n.

- **Theorem** (Garside, 1969): The conjugacy problem of B_n is solvable.

⇝ **Proposition:** For each braid b, there exists a finite, effectively computable subset $SS(b)$ of the conjugacy class of b — "summit set" of b — s.t. b, b' are conjugate iff $SS(b') = SS(b)$.

- **In practice:** $SS(b)$ is very large (exponential in the size of b),

  but improvements: ElRifai–Morton, Gonzalez-Meneses, Gebhardt,...

  ⇝ replace $SS(b)$ with smaller subsets $SSS(b)$, then $USS(b)$...

  that can be computed more easily

  ↑

  instead of considering all conjugates,
  restrict to those for which some parameter
  — the canonical length — decreases

- Garside's fundamental braid Δ_n: the half-turn on n strands

- Garside's fundamental braid Δ_n: the half-turn on n strands

- For b in B_n, define

$$\inf b = \max\{\, k \,;\, \Delta_n^k \preccurlyeq b \,\},$$

$a \preccurlyeq b$ means $b \in a \cdot x$ with no σ_i^{-1} in x

- Garside's fundamental braid Δ_n: the half-turn on n strands

- For b in B_n, define

$$\inf b = \max\{\, k \,;\, \Delta_n^k \preccurlyeq b \,\}, \qquad \sup b = \min\{\, \ell \,;\, b \preccurlyeq \Delta_n^\ell \,\},$$

$a \preccurlyeq b$ means $b \in a \cdot x$ with no σ_i^{-1} in x

- Garside's fundamental braid Δ_n: the half-turn on n strands

- For b in B_n, define

$$\inf b = \max\{\, k \,;\, \Delta_n^k \preccurlyeq b \,\}, \qquad \sup b = \min\{\, \ell \,;\, b \preccurlyeq \Delta_n^\ell \,\},$$

$a \preccurlyeq b$ means $b \in a \cdot x$ with no σ_i^{-1} in x

$\ell(b) := \sup(b) - \inf(b)$: the **canonical length** of b.

- Garside's fundamental braid Δ_n: the half-turn on n strands

- For b in B_n, define

$$\inf b = \max\{\, k \,;\, \Delta_n^k \preccurlyeq b \,\}, \qquad \sup b = \min\{\, \ell \,;\, b \preccurlyeq \Delta_n^\ell \,\},$$

$a \preccurlyeq b$ means $b \in a \cdot x$ with no σ_i^{-1} in x

$\ell(b) := \sup(b) - \inf(b)$: the **canonical length** of b.

⇝ Definition: The **SSS** of b consists of all conjugates of b with minimum canonical length.

- Garside's fundamental braid Δ_n: the half-turn on n strands

- For b in B_n, define

$$\inf b = \max\{\, k \,;\, \Delta_n^k \preccurlyeq b \,\}, \qquad \sup b = \min\{\, \ell \,;\, b \preccurlyeq \Delta_n^\ell \,\},$$

$a \preccurlyeq b$ means $b \in a \cdot x$ with no σ_i^{-1} in x

$\ell(b) := \sup(b) - \inf(b)$: the canonical length of b.

⇝ Definition: The SSS of b consists of all conjugates of b with minimum canonical length.

- We always have

$$\inf(a) + \inf(b) \leqslant \inf(ab) \leqslant \inf(a) + \sup(b),$$

- Garside's fundamental braid Δ_n: the half-turn on n strands

- For b in B_n, define

$$\inf b = \max\{\, k \,;\, \Delta_n^k \preccurlyeq b \,\}, \qquad \sup b = \min\{\, \ell \,;\, b \preccurlyeq \Delta_n^\ell \,\},$$

$a \preccurlyeq b$ means $b \in a \cdot x$ with no σ_i^{-1} in x

$\ell(b) := \sup(b) - \inf(b)$: the canonical length of b.

⇝ Definition: The SSS of b consists of all conjugates of b with minimum canonical length.

- We always have

$$\inf(a) + \inf(b) \leqslant \inf(ab) \leqslant \inf(a) + \sup(b),$$

and we almost always have

$$\inf(ab) = \inf(a) + \inf(b),$$

- Garside's fundamental braid Δ_n: the half-turn on n strands

- For b in B_n, define

$$\inf b = \max\{\, k \,;\, \Delta_n^k \preccurlyeq b \,\}, \qquad \sup b = \min\{\, \ell \,;\, b \preccurlyeq \Delta_n^\ell \,\},$$

$a \preccurlyeq b$ means $b \in a \cdot x$ with no σ_i^{-1} in x

$\ell(b) := \sup(b) - \inf(b)$: the canonical length of b.

⇝ Definition: The SSS of b consists of all conjugates of b with minimum canonical length.

- We always have

$$\inf(a) + \inf(b) \leqslant \inf(ab) \leqslant \inf(a) + \sup(b),$$

and we almost always have

$$\inf(ab) = \inf(a) + \inf(b),$$

and *id.* for $\sup$. Using $\inf(a^{-1}) = -\sup(a)$, we deduce a. a.

$$\inf(sps^{-1}) = \inf(s) + \inf(p) - \sup(s),$$

and *id.* for $\sup$, whence, a.a.,

- Garside's fundamental braid Δ_n: the half-turn on n strands

- For b in B_n, define

$$\inf b = \max\{\, k \,;\, \Delta_n^k \preccurlyeq b \,\}, \qquad \sup b = \min\{\, \ell \,;\, b \preccurlyeq \Delta_n^\ell \,\},$$

$a \preccurlyeq b$ means $b \in a \cdot x$ with no σ_i^{-1} in x

$\ell(b) := \sup(b) - \inf(b)$: the canonical length of b.

⇝ Definition: The SSS of b consists of all conjugates of b with minimum canonical length.

- We always have

$$\inf(a) + \inf(b) \leqslant \inf(ab) \leqslant \inf(a) + \sup(b),$$

and we almost always have

$$\inf(ab) = \inf(a) + \inf(b),$$

and *id.* for $\sup$. Using $\inf(a^{-1}) = -\sup(a)$, we deduce a. a.

$$\inf(sps^{-1}) = \inf(s) + \inf(p) - \sup(s),$$

and *id.* for $\sup$, whence, a.a.,

$$\ell(sps^{-1}) = \ell(p) + 2\ell(s).$$

- **Attack to the braid CSP (Hofheinz–Steinwandt '03):**

Starting with (p, q) **s.t.** p, q **are conjugate and** $\ell(p) < \ell(q)$ **:**

- **Attack to the braid CSP (Hofheinz–Steinwandt '03):**

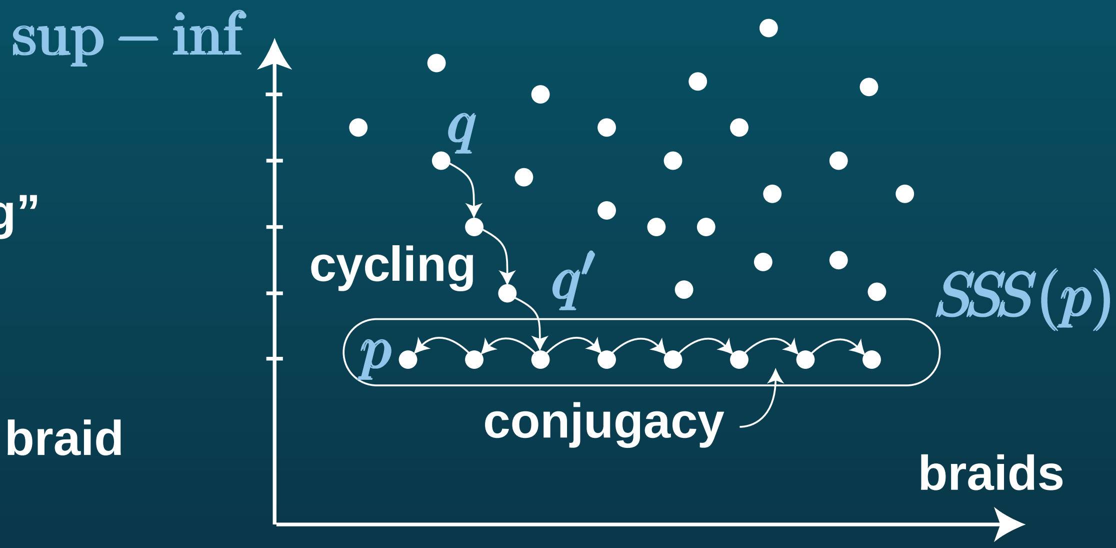
Starting with (p, q) s.t. p, q are conjugate and $\ell(p) < \ell(q)$:

- **Check $p \in SSS(p)$;**
- **Iteratively conjugate q by "cycling"**
  to increase $\inf$ & decrease $\sup$
  until $q' \in SSS(p)$;
- **Conjugate q' by one permutation braid**
  to (hopefully) obtain p.

- **Attack to the braid CSP (Hofheinz–Steinwandt '03):**

Starting with (p, q) s.t. p, q are conjugate and $\ell(p) < \ell(q)$:

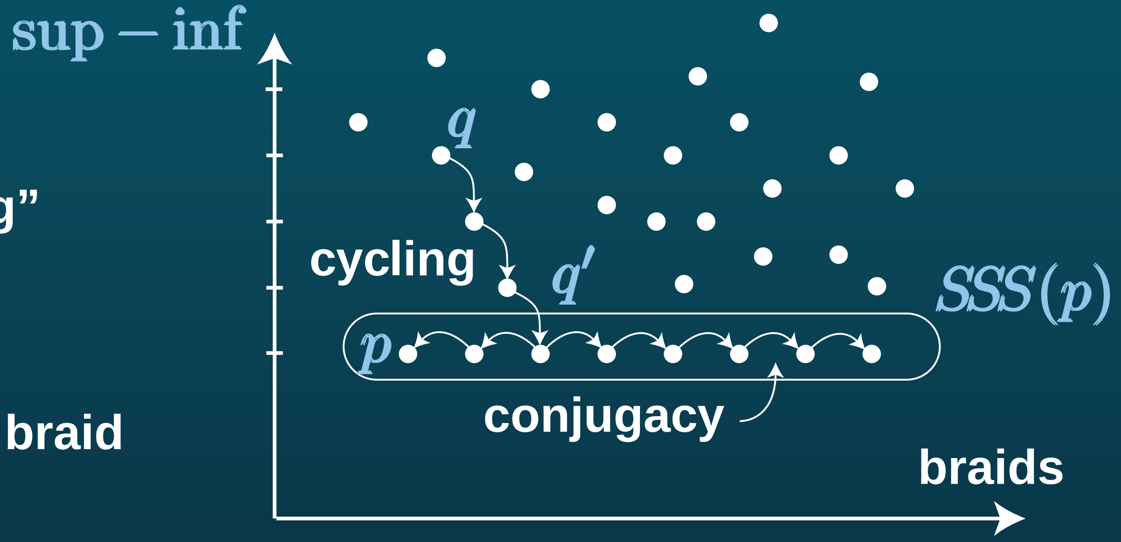
- **Check $p \in SSS(p)$;**
- **Iteratively conjugate q by "cycling"**
  to increase $\inf$ & decrease $\sup$
  until $q' \in SSS(p)$;
- **Conjugate q' by one permutation braid**
  to (hopefully) obtain p.



- **Attack to the braid CSP (Hofheinz–Steinwandt '03):**

Starting with (p, q) s.t. p, q are conjugate and $\ell(p) < \ell(q)$:

- **Check $p \in SSS(p)$;**
- **Iteratively conjugate q by "cycling" to increase $\inf$ & decrease $\sup$ until $q' \in SSS(p)$;**
- **Conjugate q' by one permutation braid to (hopefully) obtain p.**



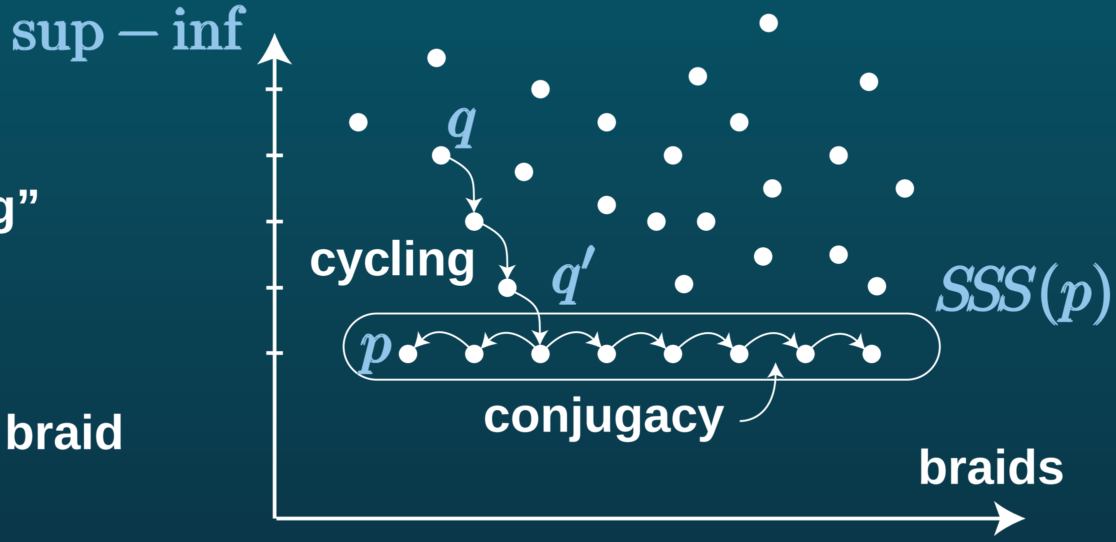
- **Key point: The attack need not always work, but it does with non-negligible probability,**

  ⇝ typically for $p \in SSS(p)$ and q obtained by conjugating p — which is frequent.

- **Attack to the braid CSP (Hofheinz–Steinwandt '03):**

Starting with (p, q) s.t. p, q are conjugate and $\ell(p) < \ell(q)$:

- Check $p \in SSS(p)$;
- Iteratively conjugate q by "cycling" to increase $\inf$ & decrease $\sup$ until $q' \in SSS(p)$;
- Conjugate q' by one permutation braid to (hopefully) obtain p.



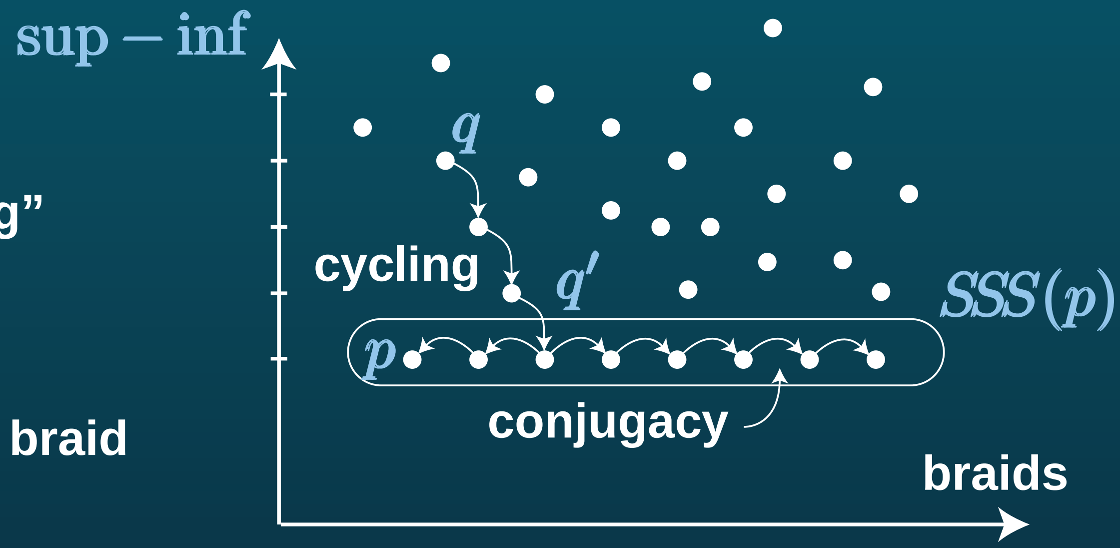
- **Key point**: The attack need not always work, but it does with **non-negligible** probability,
  ⇝ typically for $p \in SSS(p)$ and q obtained by conjugating p — which is frequent.

- **Difference between**
  - what is **mathematically** significant: what is always true,
  - what is **cryptographically** significant: what is possibly (e.g., almost always) true.

- Attack to the braid CSP (Hofheinz–Steinwandt '03):

Starting with (p, q) s.t. p, q are conjugate and $\ell(p) < \ell(q)$:

- Check $p \in SSS(p)$;
- Iteratively conjugate q by "cycling" to increase $\inf$ & decrease $\sup$ until $q' \in SSS(p)$;
- Conjugate q' by one permutation braid to (hopefully) obtain p.



- **Key point**: The attack need not always work, but it does with **non-negligible** probability,

  ⇝ typically for $p \in SSS(p)$ and q obtained by conjugating p — which is frequent.

- Difference between
  - what is **mathematically** significant: what is always true,
  - what is **cryptographically** significant: what is possibly (e.g., almost always) true.

⇝ Here: q conjugate of p implies $\ell(q) > \ell(p)$ "a.a." — although "conjugate" is symmetric...

- **Easy solution:**

  - Use conjugates in the SSS;

⇝ **Easy solution:**

- Use conjugates in the SSS;

- Do not publish (p, sps^{-1}), but $(sbs^{-1}, s'bs'^{-1})$ with b secret and s' resembling s:

same $\inf, \sup$...

⇝ **Easy solution:**

- Use conjugates in the SSS;

- Do not publish (p, sps^{-1}), but $(sbs^{-1}, s'bs'^{-1})$ with b secret and s' resembling s:

same $\inf, \sup$...

● **Authentication Protocol (Sibert '03):**

- Keys: private: b, s_0, s_1 in B_n: only A knows it; public: (p_0, p_1), with $p_i = s_i b s_i^{-1}$;

⇝ **Easy solution:**

- Use conjugates in the SSS;
- Do not publish (p, sps^{-1}), but $(sbs^{-1}, s'bs'^{-1})$ with b secret and s' resembling s: same $\inf, \sup$...

- **Authentication Protocol (Sibert '03):**

- Keys: private: b, s_0, s_1 in B_n: only A knows it; public: (p_0, p_1), with $p_i = s_i b s_i^{-1}$;
  - Repeat k times the sequence:

⇝ **Easy solution:**

- Use conjugates in the SSS;
- Do not publish (p, sps^{-1}), but $(sbs^{-1}, s'bs'^{-1})$ with b secret and s' resembling s: same $\inf, \sup$...

● **Authentication Protocol (Sibert '03):**

- Keys: private: b, s_0, s_1 in B_n: only A knows it; public: (p_0, p_1), with $p_i = s_i b s_i^{-1}$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitment $x = sbs^{-1}$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;

⇝ **Easy solution:**

- Use conjugates in the SSS;
- Do not publish (p, sps^{-1}), but $(sbs^{-1}, s'bs'^{-1})$ with b secret and s' resembling s: same $\inf, \sup$...

● **Authentication Protocol (Sibert '03):**

- Keys: private: b, s_0, s_1 in B_n: only A knows it; public: (p_0, p_1), with $p_i = s_i b s_i^{-1}$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitment $x = sbs^{-1}$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;
    - (iii) case $c = 0$
      - A sends $y = ss_0^{-1}$;
      - B checks $x = yp_0y^{-1}$;

      case $c = 1$
      - A sends $y = ss_1^{-1}$;
      - B checks $x = yp_1y^{-1}$.

⇝ **Easy solution:**

- Use conjugates in the SSS;
- Do not publish (p, sps^{-1}), but $(sbs^{-1}, s'bs'^{-1})$ with b secret and s' resembling s: same $\inf, \sup$...

● **Authentication Protocol** (Sibert '03):

- Keys: private: b, s_0, s_1 in B_n: only A knows it; public: (p_0, p_1), with $p_i = s_i b s_i^{-1}$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitment $x = sbs^{-1}$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;
    - (iii) case $c = 0$
      - A sends $y = ss_0^{-1}$;
      - B checks $x = yp_0y^{-1}$;

      case $c = 1$
      - A sends $y = ss_1^{-1}$;
      - B checks $x = yp_1y^{-1}$.

● In theory, no change; in practice, resists; uses same problem (CSP), but different instances.

⇝ **Easy solution:**

- Use conjugates in the SSS;
- Do not publish (p, sps^{-1}), but $(sbs^{-1}, s'bs'^{-1})$ with b secret and s' resembling s: same $\inf, \sup$...

- **Authentication Protocol** (Sibert '03):

- Keys: private: b, s_0, s_1 in B_n: only A knows it; public: (p_0, p_1), with $p_i = s_i b s_i^{-1}$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitment $x = sbs^{-1}$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;
    - (iii) case $c = 0$: A sends $y = ss_0^{-1}$; B checks $x = yp_0y^{-1}$;
      case $c = 1$: A sends $y = ss_1^{-1}$; B checks $x = yp_1y^{-1}$.

- In theory, no change; in practice, resists; uses same problem (CSP), but different instances.

⇝ main problem: choosing the instances (cf. RSA...)

- **Should one renounce to braid cryptography?**

- **Should one renounce to braid cryptography? NO ⇝ just work on it!**

- Should one renounce to braid cryptography? NO ⇝ just work on it!
  ⇝ several options:

- Option 1 : Keep the conjugacy problem as the primitive, but choose the keys better;
  ⇝ Find families of braids with large SSS and USS (Ko, Lee);
    ⇝ connected with dynamical properties and the Nielsen–Thurston theory;
    ⇝ also depends on the way braids are specified (normal form *vs.* arbitrary words).

- Should one renounce to braid cryptography? NO ⇝ just work on it!
  ⇝ several options:

- Option 1 : Keep the conjugacy problem as the primitive, but choose the keys better;
  - ⇝ Find families of braids with large SSS and USS (Ko, Lee);
    - ⇝ connected with dynamical properties and the Nielsen–Thurston theory;
    - ⇝ also depends on the way braids are specified (normal form *vs.* arbitrary words).

- Option 2: Use a new primitive such as the root problem: starting from p, find s s.t. $s^2 = p$.
  - ⇝ connected with the conjugacy problem, and solvable in exponential time (Stychnev).

- Should one renounce to braid cryptography? NO ⇝ just work on it!

  ⇝ several options:

- Option 1 : Keep the conjugacy problem as the primitive, but choose the keys better;
  - ⇝ Find families of braids with large SSS and USS (Ko, Lee);
    - ⇝ connected with dynamical properties and the Nielsen–Thurston theory;
    - ⇝ also depends on the way braids are specified (normal form *vs.* arbitrary words).

- Option 2: Use a new primitive such as the root problem: starting from p, find s s.t. $s^2 = p$.
  - ⇝ connected with the conjugacy problem, and solvable in exponential time (Stychnev).

- Option 3: Use a really new primitive such as the shifted conjugacy problem:

  Replace

$$s * p = s \cdot p \cdot s^{-1}$$

- Should one renounce to braid cryptography? NO ⇝ just work on it!

  ⇝ several options:

- Option 1 : Keep the conjugacy problem as the primitive, but choose the keys better;
  - ⇝ Find families of braids with large SSS and USS (Ko, Lee);
    - ⇝ connected with dynamical properties and the Nielsen–Thurston theory;
    - ⇝ also depends on the way braids are specified (normal form *vs.* arbitrary words).

- Option 2: Use a new primitive such as the root problem: starting from p, find s s.t. $s^2 = p$.
  - ⇝ connected with the conjugacy problem, and solvable in exponential time (Stychnev).

- Option 3: Use a really new primitive such as the shifted conjugacy problem:

  Replace

$$s * p = s \cdot p \cdot s^{-1}$$

  with

$$s * p := s \cdot \partial p \cdot \sigma_1 \cdot \partial s^{-1}$$

↑ the shift endomorphism $\sigma_i \mapsto \sigma_{i+1}$ for each i

- **Authentication protocol:**

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = s * p$;

- **Authentication protocol:**

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = s * p$;

- Repeat k times the sequence:

(i) A chooses r in B_n, and sends the commitments $x = r * p$ & $y = r * q$;

- **Authentication protocol:**

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = s * p$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitments $x = r * p$ & $y = r * q$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;

- **Authentication protocol:**

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = s * p$;

  - Repeat k times the sequence:

    (i) A chooses r in B_n, and sends the commitments $x = r * p$ & $y = r * q$;

    (ii) B chooses c in $\{0, 1\}$, and sends c;

    (iii) case $c = 0$

    A sends $z = r$;

    B checks $x = z * p$ & $y = z * q$;

- **Authentication protocol:**

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = s * p$;

- Repeat k times the sequence:

(i) A chooses r in B_n, and sends the commitments $x = r * p$ & $y = r * q$;

(ii) B chooses c in $\{0, 1\}$, and sends c;

(iii) case $c = 0$

A sends $z = r$;

B checks $x = z * p$ & $y = z * q$;

case $c = 1$

A sends $z = r * s$;

B checks $y = z * x$.

- **Authentication protocol:**

- Keys: private: s **in** B_n**: only A knows it; public:** (p, q)**, with** p **in** B_n **and** $q = s * p$**;**

- Repeat k **times the sequence:**

(i) A chooses r **in** B_n**, and sends the commitments** $x = r * p$ **&** $y = r * q$**;**

(ii) B chooses c **in** $\{0, 1\}$**, and sends** c**;**

(iii) case $c = 0$

A sends $z = r$**;**

B checks $x = z * p$ **&** $y = z * q$**;**

case $c = 1$

A sends $z = r * s$**;**

B checks $y = z * x$**.**

- **Justification (case** $c = 1$**):**

$$y = r * q = r * (s * p) = (r * s) * (r * p) = z * x,$$

- **Authentication protocol:**

- **Keys: private:** s **in** B_n**: only A knows it; public:** (p, q)**, with** p **in** B_n **and** $q = s * p$;
  - **Repeat** k **times the sequence:**
    - **(i) A chooses** r **in** B_n**, and sends the commitments** $x = r * p$ **&** $y = r * q$;
    - **(ii) B chooses** c **in** $\{0, 1\}$**, and sends** c;
    - **(iii) case** $c = 0$: **A sends** $z = r$; **B checks** $x = z * p$ **&** $y = z * q$;
    - **case** $c = 1$: **A sends** $z = r * s$; **B checks** $y = z * x$.

- **Justification (case** $c = 1$**):**

$$y = r * q = r * (s * p) = (r * s) * (r * p) = z * x,$$

↑ **the point: like conjugacy, operation** $*$ **is self-distributive**

- **Authentication protocol:**

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = s * p$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitments $x = r * p$ & $y = r * q$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;
    - (iii) case $c = 0$: A sends $z = r$; B checks $x = z * p$ & $y = z * q$;
      case $c = 1$: A sends $z = r * s$; B checks $y = z * x$.

- **Justification (case $c = 1$):**

$$y = r * q = r * (s * p) = (r * s) * (r * p) = z * x,$$

the point: like conjugacy, operation $*$ is self-distributive

- **Probability that A succeeds = probability of finding z s.t. $z * (r * p) = (r * s) * (r * p)$.**

⇝ size of the shifted commutator of $a = \partial(r * p)\sigma_1$

$$C_\partial(a) = \{x; x\,a = a\,\partial x\}$$

- **Authentication protocol:**

- Keys: private: s in B_n: only A knows it; public: (p, q), with p in B_n and $q = s * p$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitments $x = r * p$ & $y = r * q$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;
    - (iii) case $c = 0$: A sends $z = r$; B checks $x = z * p$ & $y = z * q$;
      case $c = 1$: A sends $z = r * s$; B checks $y = z * x$.

- **Justification (case $c = 1$):**

$$y = r * q = r * (s * p) = (r * s) * (r * p) = z * x,$$

↑ the point: like conjugacy, operation $*$ is self-distributive

- **Probability that A succeeds = probability of finding z s.t. $z * (r * p) = (r * s) * (r * p)$.**

  ⇝ size of the shifted commutator of $a = \partial(r * p)\sigma_1$

  ↖ $C_\partial(a) = \{x ; x\, a = a\, \partial x\}$

  ⇝ very small, *e.g.*, $C_\partial(1) = \{1\}$.

- **Authentication protocol (variant):**

- **Authentication protocol (variant):**

- Keys: private: s in B_n: only A knows it; public: p, with $p = s * s$;

- Repeat k times the sequence:

(i) A chooses r in B_n, and sends the commitment $x = r * p$;

(ii) B chooses c in $\{0, 1\}$, and sends c;

- **Authentication protocol (variant):**

- Keys: private: s in B_n: only A knows it; public: p, with $p = s * s$;

- Repeat k times the sequence:

(i) A chooses r in B_n, and sends the commitment $x = r * p$;

(ii) B chooses c in $\{0, 1\}$, and sends c;

(iii) case $c = 0$

A sends $y = r$;

B checks $x = y * p$;

- **Authentication protocol (variant):**

- Keys: private: s in B_n: only A knows it; public: p, with $p = s * s$;

- Repeat k times the sequence:

(i) A chooses r in B_n, and sends the commitment $x = r * p$;

(ii) B chooses c in $\{0, 1\}$, and sends c;

(iii) case $c = 0$

A sends $y = r$;

B checks $x = y * p$;

case $c = 1$

A sends $y = r * s$;

B checks $x = y * y$.

- **Authentication protocol (variant):**

- Keys: private: s in B_n: only A knows it; public: p, with $p = s * s$;
  - Repeat k times the sequence:
    - (i) A chooses r in B_n, and sends the commitment $x = r * p$;
    - (ii) B chooses c in $\{0, 1\}$, and sends c;
    - (iii) case $c = 0$
      - A sends $y = r$;
      - B checks $x = y * p$;
    - case $c = 1$
      - A sends $y = r * s$;
      - B checks $x = y * y$.

- **Justification (case $c = 1$):**

$$x = r * q = r * (s * s) = (r * s) * (r * s) = y * y,$$

↑ self-distributivity of $*$ again

- **Is this the future of braid cryptography?**

- **Is this the future of braid cryptography? NO $\rightsquigarrow$ just work on it!**

- **Is this the future of braid cryptography?** **NO** ⇝ **just work on it!**
  ⇝ **many questions and possibilities.**
- **How to prove security results?**

- Is this the future of braid cryptography? NO ⇝ just work on it!

  ⇝ many questions and possibilities.

- How to prove security results?
  - What is a random braid?

    ⇝ no invariant probability measure on B_n: not an amenable group;

- Is this the future of braid cryptography? **NO** $\rightsquigarrow$ just **work** on it!

  $\rightsquigarrow$ many questions and possibilities.

- How to **prove** security results?

  - What is a random braid?

    $\rightsquigarrow$ no invariant probability measure on B_n: not an amenable group;

- How to choose the keys?

- Is this the future of braid cryptography? NO ⇝ just work on it!
  ⇝ many questions and possibilities.

- How to prove security results?
  - What is a random braid?
    ⇝ no invariant probability measure on B_n: not an amenable group;

- How to choose the keys?
  - What is needed, *e.g.,* for the CSP:
    ⇝ not a proof that all instances of the problem are difficult,

- Is this the future of braid cryptography? **NO** ⇝ just **work** on it!
  ⇝ many questions and possibilities.

- How to **prove** security results?
  - What is a random braid?
    - ⇝ no invariant probability measure on B_n: not an amenable group;

- How to choose the keys?
  - What is needed, *e.g.,* for the CSP:
    - ⇝ not a proof that **all** instances of the problem are difficult,
    - ⇝ but a method for constructing **some** (enough) provably difficult instances.
      - here connected with the size of the SSS and USS;

- Is this the future of braid cryptography? NO $\rightsquigarrow$ just work on it!

  $\rightsquigarrow$ many questions and possibilities.

- How to prove security results?
  - What is a random braid?
    - $\rightsquigarrow$ no invariant probability measure on B_n: not an amenable group;

- How to choose the keys?
  - What is needed, *e.g.,* for the CSP:
    - $\rightsquigarrow$ not a proof that all instances of the problem are difficult,
    - $\rightsquigarrow$ but a method for constructing some (enough) provably difficult instances.
      - here connected with the size of the SSS and USS;
  - *Id.* for the SCSP

    ↖ Shifted Conjugacy Search Problem

- **Is this the future of braid cryptography? NO ⇝ just work on it!**
  ⇝ many questions and possibilities.

- **How to prove security results?**
  - **What is a random braid?**
    - **⇝ no invariant probability measure on B_n: not an amenable group;**

- **How to choose the keys?**
  - **What is needed, *e.g.,* for the CSP:**
    - **⇝ not a proof that all instances of the problem are difficult,**
    - **⇝ but a method for constructing some (enough) provably difficult instances.**
      - **here connected with the size of the SSS and USS;**
  - ***Id.* for the SCSP (easier because SCSP (much) more difficult than CSP).**

Shifted Conjugacy Search Problem

- Is this the future of braid cryptography? NO $\rightsquigarrow$ just work on it!
  $\rightsquigarrow$ many questions and possibilities.

- How to prove security results?
  - What is a random braid?
    $\rightsquigarrow$ no invariant probability measure on B_n: not an amenable group;

- How to choose the keys?
  - What is needed, *e.g.*, for the CSP:
    $\rightsquigarrow$ not a proof that all instances of the problem are difficult,
    $\rightsquigarrow$ but a method for constructing some (enough) provably difficult instances.
      - here connected with the size of the SSS and USS;
  - *Id.* for the SCSP (easier because SCSP (much) more difficult than CSP).
    ↖ Shifted Conjugacy Search Problem

- Use Dynnikov's formulas, in particular to design hash functions.
  ↖ coordinization map $B_n \to \mathbb{Z}^{2n}$ coming from the theory of laminations

- Is this the future of braid cryptography? NO $\rightsquigarrow$ just work on it!

  $\rightsquigarrow$ many questions and possibilities.

- How to prove security results?
  - What is a random braid?
    - $\rightsquigarrow$ no invariant probability measure on B_n: not an amenable group;

- How to choose the keys?
  - What is needed, *e.g.*, for the CSP:
    - $\rightsquigarrow$ not a proof that all instances of the problem are difficult,
    - $\rightsquigarrow$ but a method for constructing some (enough) provably difficult instances.
      - here connected with the size of the SSS and USS;
  - *Id.* for the SCSP (easier because SCSP (much) more difficult than CSP).

    ↖ Shifted Conjugacy Search Problem

- Use Dynnikov's formulas, in particular to design hash functions.

  ↖ coordinization map $B_n \to \mathbb{Z}^{2n}$ coming from the theory of laminations

... and much more still to be discovered.