Thus far, we have assumed that parties have a shared key. Where does the shared key come from?

**Approach 1:** have a key-distribution center (KDC)

- **Shared key between KDC and each party Pi:**
  - if Pi wants to talk to Pj:
    - Pi sends nonce r_i, (replay protection) and identifier id_i to Pj
    - Pj chooses nonce r_j and identifier id_j to Pi and KDC
    - KDC samples k_{ij} and gives
      - nonce needed to ensure "freshness" for session (no replay)
      - identifiers needed to bind session key k_{ij} to identities id_i, id_j

A “ticket”:

- \( c_i \leftarrow \text{Enc}(k_i, \text{Enc}, k_{ij}) \) to Pi
- \( t_i \leftarrow \text{MAC}(k_i, \text{mac}, (r_i, r_j, id_i, id_j, c_i)) \) to Pj
- \( c_j \leftarrow \text{Enc}(k_j, \text{Enc}, k_{ij}) \)
- \( t_j \leftarrow \text{MAC}(k_j, \text{mac}, (r_i, r_j, id_i, id_j, c_j)) \) to Pj

Basic design for Kerberos - only requires symmetric primitives

- **Drawback:** KDC must be fully trusted (knows everyone’s keys) and is single point of failure (no session setup if KDC goes offline)
Public-key cryptography: Session setup / key exchange without a KDC

Diffie-Hellman key exchange (example) — will be more precise later:

- Assume we have a fixed prime \( p \) and a value \( g \in \{1, 2, \ldots, p-1\} \) (these could be specified in a cryptographic standard)

\[
\begin{align*}
\text{Alice} & \quad X \in \{1, 2, \ldots, p-1\} \\
\text{Bob} & \quad Y \in \{1, 2, \ldots, p-1\}
\end{align*}
\]

\[
\begin{array}{c}
X \xrightarrow{\text{computes}} g^X \pmod{p} \\
Y \xrightarrow{\text{computes}} g^Y \pmod{p}
\end{array}
\]

- Alice computes \( g^Y \pmod{p} \)
- Bob computes \( g^X \pmod{p} \)

\[
\begin{array}{c}
g^X = (g^Y)^X \pmod{p} \\
g^Y = (g^X)^Y \pmod{p}
\end{array}
\]

\[
\text{shared secret: } g^Y
\]

- Assumption: given only \((g, p), g^X, g^Y\), it is difficult to compute \( g^Y \) [computational Diffie-Hellman assumption]
- To better be the case that computing logarithms base \( g \) be difficult [discrete logarithm problem]

(e.g., given \( g, g^X \), cannot compute \( X \))

To understand this more broadly, we will need some math background. We discuss some key facts from number theory and abstract algebra below:

Definition. A group consists of a set \( G \) together with an operation \( * \) that satisfies the following properties:
- Closure: If \( g_1, g_2 \in G \), then \( g_1 * g_2 \in G \)
- Associativity: For all \( g_1, g_2, g_3 \in G \), \( g_1 * (g_2 * g_3) = (g_1 * g_2) * g_3 \)
- Identity: There exists an element \( e \in G \) such that \( e * g = g = g * e \) for all \( g \in G \)
- Inverse: For every element \( g \in G \), there exists an element \( g^{-1} \in G \) such that \( g * g^{-1} = e = g^{-1} * g \)

In addition, we say a group is commutative (or abelian) if the following property also holds:
- Commutativity: For all \( g_1, g_2 \in G \), \( g_1 * g_2 = g_2 * g_1 \)

[ called "multiplicative" notation ]

Notation: Typically, we will use "\( \cdot \)" to denote the group operation (unless explicitly specified otherwise). We will write \( g^k \) to denote \( g \cdot g \cdot g \cdots g \) (the usual exponential notation). We use "1" to denote the multiplicative identity.

\( x \) times

Examples of groups:
- \( \mathbb{R}, + \): real numbers under addition
- \( \mathbb{Z}, + \): integers under addition
- \( \mathbb{Z}_p, + \): integers modulo \( p \) under addition [sometimes written as \( \mathbb{Z}/p\mathbb{Z} \)]

\( \text{here, } p \text{ is prime } \)

The structure of \( \mathbb{Z}^* \) (an important group for cryptography):
\[
\mathbb{Z}^*_p = \{ x \in \mathbb{Z}_p : \text{there exists } y \in \mathbb{Z}_p \text{ where } xy = 1 \pmod{p} \}
\]
\( \text{the set of elements with multiplicative inverses modulo } p \)