Security in WebRTC Peer-To-Peer connections and knowing who you're talking to

Marc Matija

Hochschule RheinMain

2024-01-05

Introduction

- Surge in applications like video conferencing, online collaboration, multiplayer gaming, and content sharing.
- Growing reliance on P2P technologies like WebRTC for direct, real-time data exchange.

source: https://commons.wikimedia.org/

What is ...

?

- A web standard enabling audio, video, and data transmission directly between browsers or clients without central servers.
- Enables Cross-Platform Peer-To-Peer connection
- Adopted in platforms like Big Blue Button, Zoom, and Discord.

source: https://commons.wikimedia.org/wiki/File:WebRTC_Logo.svg

Why use Peer-To-Peer?

- **Decentralization**: No need for a central server, reducing single points of failure and increasing reliability.
- **Reduced Latency**: Direct connections between peers minimize delays compared to routing through a server.
- **Scalability**: As the number of users grows, each new peer contributes to the network’s resources, enabling better scalability.
- **Cost Efficiency**: Reduces infrastructure costs by distributing the load across users, rather than relying on expensive server farms.

Objectives

Presentation Goals

- Understand WebRTC’s architecture and security challenges.
- Understand how these challenges are solved.
- Highlight the risks involved with peer-to-peer.
- Explore security mechanisms and mitigation strategies.

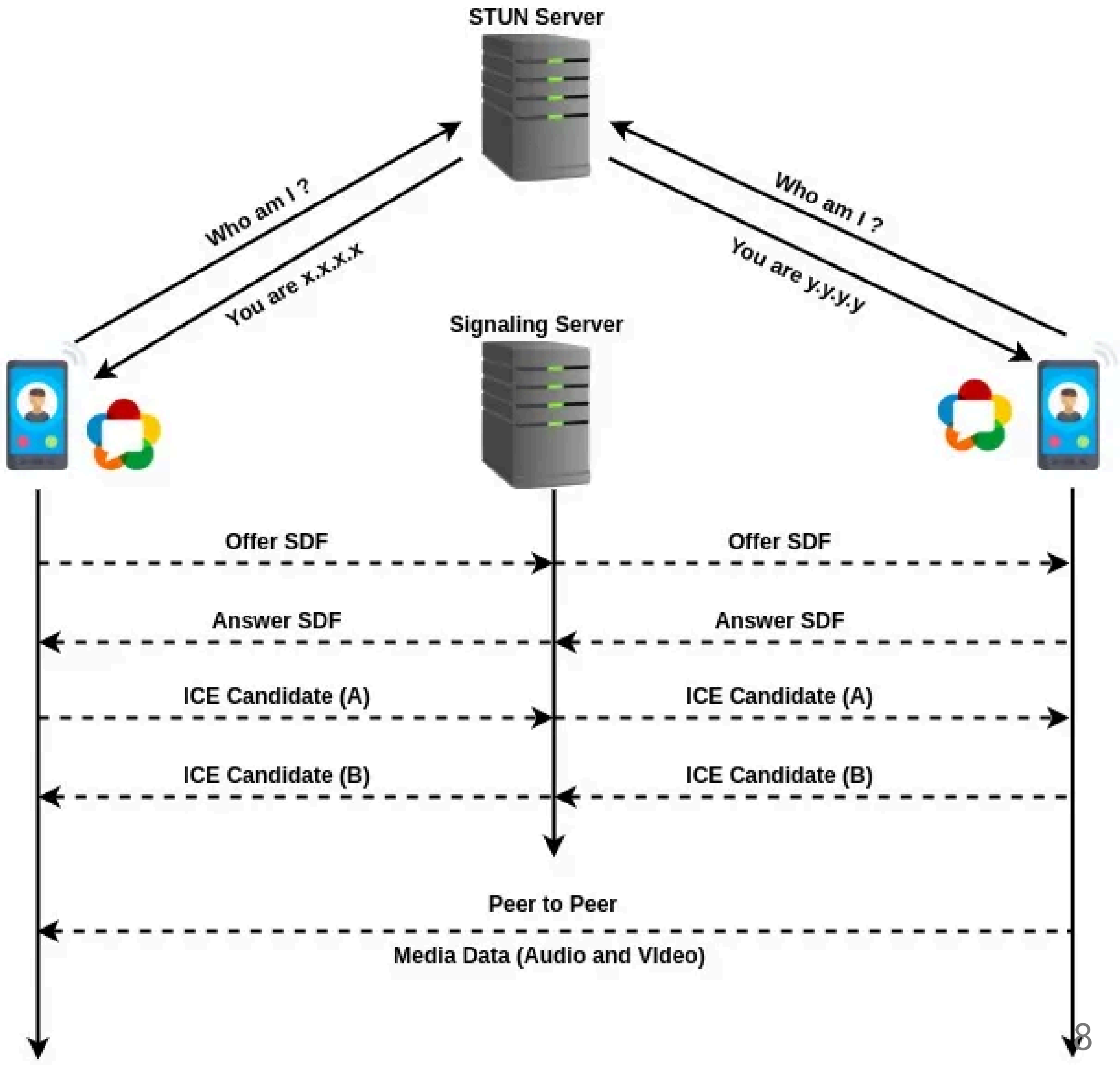
Core Components

- **NAT (Network Address Translation):** Method of mapping an IP address space into another by modifying network address information in the IP header of packets.
- **SDP (Session Description Protocol):** Describes session information.
- **STUN (Session Traversal Utilities for NAT):** Discovers public IP and port.
- **TURN (Traversal Using Relays around NAT):** Relays traffic when direct connections fail.
- **ICE (Interactive Connectivity Establishment):** Selects the best connection path.
- **DTLS (Datagram Transport Layer Security):** Encrypts data streams.
- **SRTP (Secure Real-time Transport Protocol):** Secures media streams.

How WebRTC Works

- **Signaling**
    - Exchanges SDP messages and ICE candidates.
- **NAT Traversal**
    - Uses STUN and TURN for connectivity.
- **Connection Establishment**
    - UDP as underlying communications Protocol
    - DTLS handshake for encryption.
    - SRTP/SCTP for secure media and data transmission.

- Find public IP and port using STUN.
- Exchange SDP (offer/answer)
- Exchange ICE candidates.
- Connect to Peer
  - Use TURN if direct connection fails



source: https://commons.wikimedia.org/wiki/File:WebRTC_Logo.svg

How to know, who to trust?

- Lack of central authority
- Anonymous nature of WebRTC
- Impersonation risks

source: https://michaelmaslin.com/peter-steiner-20th-anniversary-on-internet-nobody-knows-youre-dog-2/

"On the Internet, nobody knows you're a dog."

The Trust Model in WebRTC

- **Decentralized Trust Model:**
  - WebRTC lacks a central authority to verify peers.
  - Relies on the signaling server and cryptographic protocols.
- **Goal:**
  - Ensure the integrity, authenticity, and confidentiality of communication.

Authenticated Entities

- **Identity Providers (IdPs):**
  - Provide credentials to verify the identity of peers.
  - Use tokens to establish mutual trust.
- **Signaling Server Trust:**
  - Facilitates SDP and ICE exchange.
  - Does not participate in media transmission.
  - Secure signaling via HTTPS or WSS (WebSockets).

Unauthenticated Entities

- **Challenges of Unauthenticated Entities:**
  - No direct verification of peer identity in many cases.
  - Vulnerable to impersonation or unauthorized access.
- **Examples:**
  - Public peer-to-peer gaming lobbies.
  - Ad hoc WebRTC-based communication tools.
- **Mitigation Strategies:**
  - Use of tokens or passphrases for peer verification.
  - Regularly monitor TURN server activity.
  - Implement rate-limiting to prevent abuse.

Authentication ≠ Trust

- **Verifying identity** (e.g., Dr. Evil owns `example.org`) does not imply trustworthiness.
- **User Decision:** Users must decide whether to grant access based on the authenticated entity.
- **Temporary Trust:** Access to sensitive resources (e.g., camera/mic) should be *limited to context-specific use* (e.g., a single call).
- **Identification as Prerequisite for Trust:** Policies depend on proper identification of network elements. Identification enables informed trust decisions and policy application.

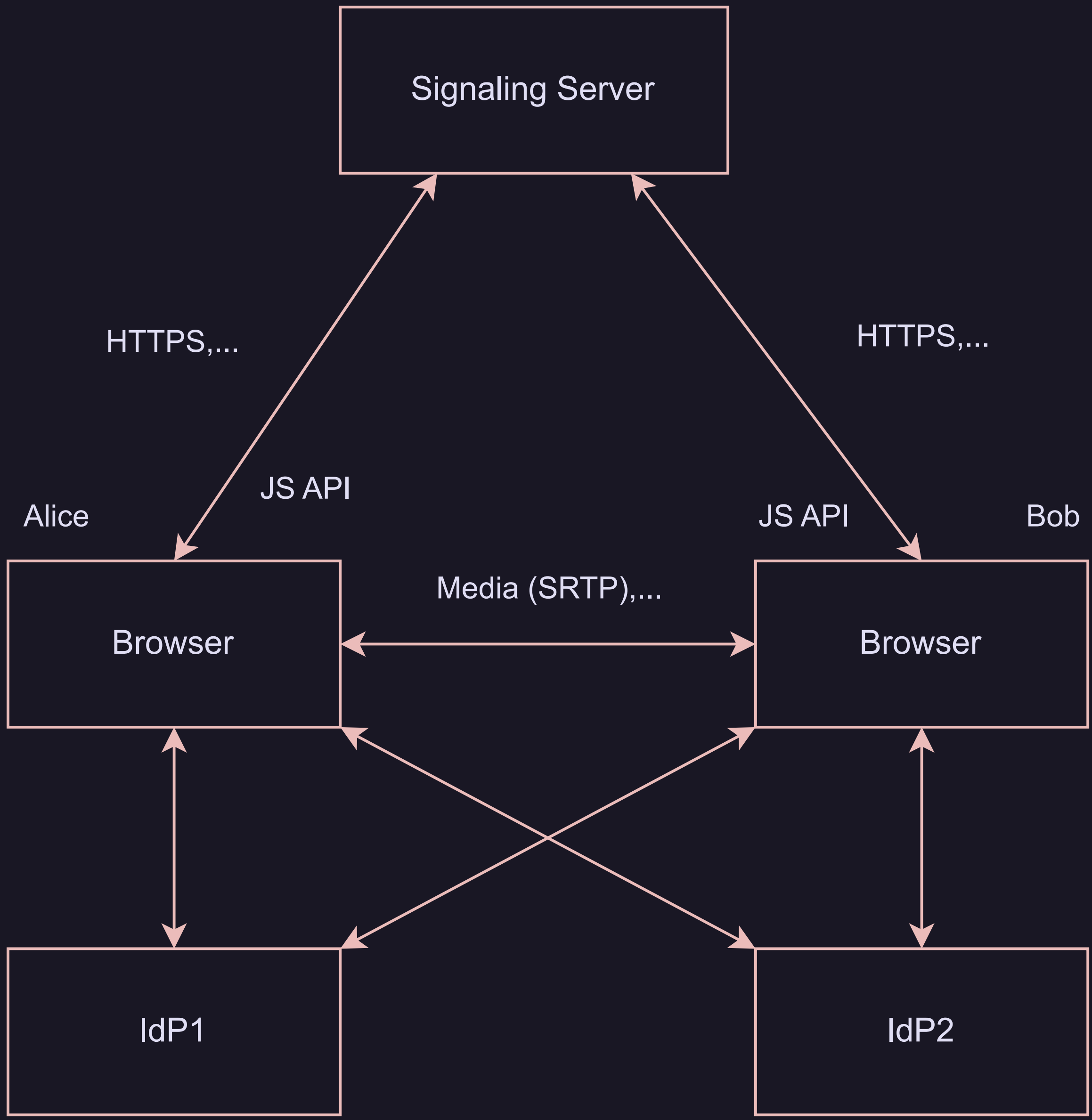
Identification

Challenges in Identification

- **Dynamic Networks**: NATs and changing IPs hinder peer identification.
- **No Central Authority**: Decentralized model lacks built-in verification.
- **Impersonation Risks**: Malicious entities can mimic legitimate peers.
- **Context Matters**: Trust varies across use cases (e.g., public vs. private).
- **Security vs. Usability**: Balancing ease of use with strong identity checks.

Role of Identity Providers (IdPs)

- Identity Verification: Cryptographic credentials ensure authenticity.
- Token-based Trust: Tokens exchanged for secure signaling.
- User Convenience: Trusted logins simplify identity management.
- Supports Decentralization: Maintains WebRTC's P2P architecture.
- Enables Trust Decisions: Verified identities guide user trust.



source: https://www.rfc-editor.org/rfc/rfc8827.html

Authenticity and Data Integrity

- WebRTC uses UDP as it's default protocol with TCP as a fallback
  - UDP being a connectionless Protocol brings it's own challenges
- End-to-End encryption by default.
- Protocols used for data security:
  - **DTLS**
  - **SRTP**

DTLS

- **Encryption:** Ensures that data transmitted between peers is encrypted to prevent unauthorized access. Uses symmetric encryption (e.g., AES, ChaCha20) for fast and secure communication.
- **Authentication:** Verifies the identity of both parties during the handshake process. Uses digital certificates or pre-shared keys (PSKs).
- **Integrity:** Protects against data tampering by using cryptographic hash functions (e.g., HMAC).
- **Replay Protection:** Prevents attackers from re-sending captured packets by assigning sequence numbers and timeouts.

SRTP (Secure Real-time Transport Protocol)

- **Encryption for Confidentiality**: Encrypts media content using AES in Counter Mode.
- **Message Authentication**: Uses HMAC-SHA1 to verify integrity and prevent tampering.
- **Replay Protection**: Includes sequence numbers to prevent replay attacks.
- Relies on DTLS for exchanging cryptographic keys during the WebRTC handshake

Potential Risks of WebRTC

- WebRTC Leaks
- Man-In-The-Middle
- Exploitation of Vulnerable TURN Servers

WebRTC Leaks

- **IP Address Leakage**: WebRTC requires knowledge of a peer's IP address, potentially exposing the user's location.
- **Risk of Location Exposure**: Even when using VPNs, WebRTC can leak:
    - Public IPv6 addresses
    - Temporary IPv6 addresses
    - Local and private addresses
- **Privacy Concerns**: VPNs aim to mitigate this, but WebRTC still leaks IP addresses in some scenarios.

Mitigating Risks of WebRTC Leaks

- **Disabling WebRTC**: Turning off WebRTC in the browser to prevent leaks.
- **Disabling IPv6**: Reducing the scope of leaked information by disabling IPv6.
- **Using Relay Servers**: Sending data through a central server, though it negates the peer-to-peer nature of WebRTC.
- **No Perfect Solution**: Despite mitigations, WebRTC still inherently leaks IP addresses, sometimes even with VPN protection.

Man-In-The-Middle Attacks in WebRTC

- **Vulnerable Signaling Process**: WebRTC requires signaling to establish connections, which can be intercepted by attackers.
- **Key Substitution & Impersonation**: Attackers can replace cryptographic fingerprints, tricking peers into connecting with the attacker instead of the intended peer.
- **Session Hijacking**: Attackers can modify SDP parameters to redirect traffic through malicious servers or hijack communication sessions.

Mitigating MITM Attacks

- **Eavesdropping on Signaling**: Without encryption, attackers can intercept metadata like SDP messages, ICE candidates, and DTLS fingerprints.
- **Encrypted Signaling**: Protect signaling channels with encryption to prevent interception and manipulation.
- **Monitoring Media Path**: Regular checks for suspicious relays to detect MITM activity and secure the communication.

Exploitation of Vulnerable TURN Servers

- **Bandwidth Drain Attacks**: Misconfigured TURN servers without authentication were used to relay high-volume traffic, leading to financial losses.
- **Abuse in Botnets**: Vulnerable servers were exploited to create resilient command-and-control (C&C) infrastructures, bypassing firewalls and NAT restrictions.
- **Sensitive Media Interception**: Lack of encryption on TURN servers allowed attackers to eavesdrop on real-time communications, especially on public cloud infrastructure.
- **Case Example – Slack’s Misconfigured TURN Servers**: Attackers exploited weak authentication to gain unauthorized access to internal services, bypassing network restrictions.

Mitigating Risks of Vulnerable TURN Servers

- **Enforce Authentication**: Use strong authentication mechanisms, such as long-term credentials or OAuth tokens, with short expiration times.
- **Restrict Access**: Limit access to TURN servers by configuring firewalls and using rate-limiting to prevent abuse.
- **Encrypt Traffic**: Ensure TURN traffic is encrypted with protocols like DTLS or TLS.
- **Monitor Server Usage**: Regularly audit server logs for unusual patterns or unauthorized access attempts.
- **Secure Deployment**: Avoid hosting TURN servers on shared or insecure cloud environments; deploy in trusted locations.

Conclusion

- **WebRTC’s Impact**: Revolutionized real-time communication with Peer-to-Peer architecture, reducing reliance on centralized servers.
- **Security Challenges**: Includes signaling vulnerabilities, IP address leaks, and the need for trust establishment.
- **Existing Solutions**: DTLS-SRTP and Identity Providers offer strong encryption and authentication, but vulnerabilities remain.
- **Future Directions**: Continued advancements in protocol design and heightened awareness of risks are needed to secure WebRTC for the modern web.

Signaling Standards

WebRTC has no direct standard for the signaling protocol, however some popular options include

- **Extensible Messaging and Presence Protocol (XMPP)**
    - XMPP is an open-standard communication protocol that facilitates instant messaging and presence updates.
- **Session Initiation Protocol (SIP)**
    - SIP is a signaling protocol commonly used in telecommunications to establish, modify, and terminate multimedia sessions.
- **Custom REST API + WebSockets solution**

Extended Usages for WebRTC

- Real-Time Collaboration Tools
  - Example: Tools like **Google Docs** or **Miro (virtual whiteboard)** use WebRTC to synchronize changes made by users in real-time across different locations.
- Online Gaming (Real-Time Multiplayer)
  - Example: **Valve's** **GameNetworkingSockets** library uses WebRTC for Peer-To-Peer multiplayer
- File Sharing and Data Transfer
  - Eliminating the need for a centralized file server. f.e. **FilePizza**, is a peer-to-peer file-sharing platform.

Slack's vulnerable Turn Servers

Executive Summary:

- Vulnerability Identified: Slack's TURN server allowed relaying of TCP connections and UDP packets to internal Slack network and AWS meta-data services.
- Bug Bounty: $3,500 awarded for the discovery via HackerOne.
- Abuse of TURN: Slack's TURN server was used to relay traffic to:
    - AWS Meta-Data Services: Access IAM temporary credentials.
    - Internal Open Ports: Ports like 22, 25, 443, etc. on Slack's internal servers.
    - Port Scanning: Scan internal IP range (10.41.0.0/16) for management applications.

source: https://www.enablesecurity.com/blog/slack-webrtc-turn-compromise-and-bug-bounty/

NAT (Network Address Translation)

- **Definition:** NAT is a networking process that modifies the source or destination IP address of packets as they pass through a router or firewall.
- **Purpose:** Enables multiple devices on a local network (private IPs) to share a single public IP address when accessing external networks like the Internet.

How NAT Works:

- Devices in a local network are assigned private IP addresses (e.g., 192.168.x.x).
- The NAT router translates the private IP into the network's public IP when sending traffic to the Internet.
- NAT uses port numbers to keep track of which internal device corresponds to each connection.

Types of NAT:

- **Static NAT:** One-to-one mapping between private and public IP addresses.
- **Dynamic NAT:** Maps private IPs to a pool of public IPs on a first-come, first-served basis.
- **PAT (Port Address Translation):** Multiple private IPs share a single public IP, with traffic differentiated by port numbers.

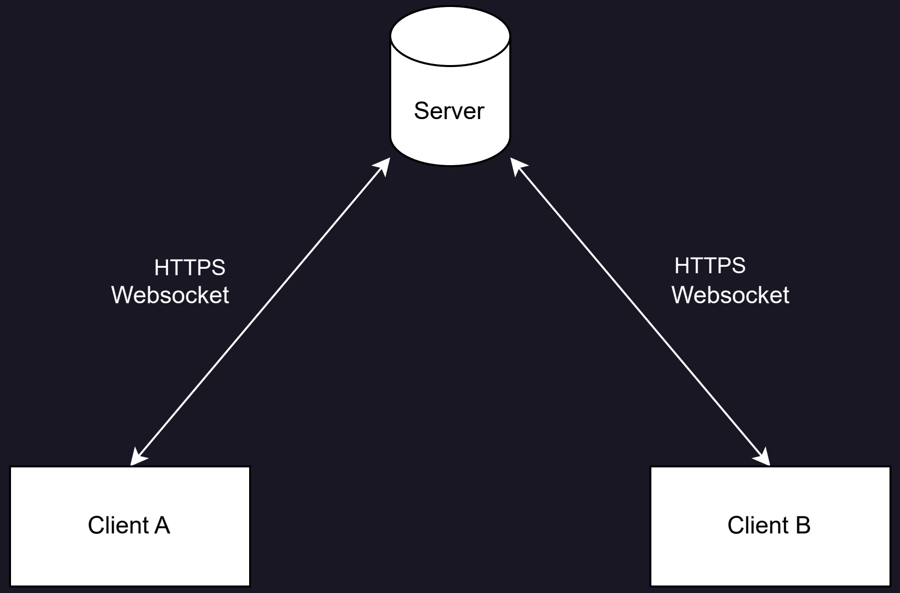
Challenges of NAT:

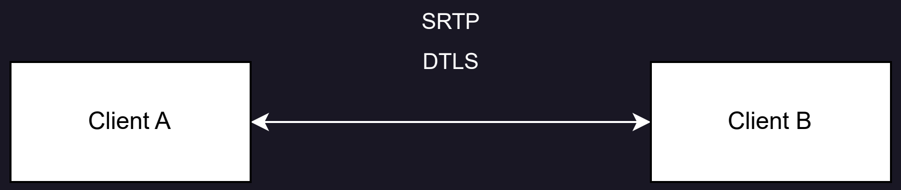
- **Breaks Peer-to-Peer (P2P) Communication:** NAT makes direct communication between devices behind different NATs difficult.
- Workarounds: Protocols like STUN, TURN, and ICE are used to traverse NAT in applications like WebRTC.

DTLS Handshake Process

- ClientHello: Initiates communication and proposes cryptographic algorithms.
- ServerHello: Server responds with selected algorithms and its certificate.
- Key Exchange: Both parties securely exchange keys using asymmetric encryption.
- Session Keys: Generate shared session keys for encrypting communication.

Differences Between Peer-To-Peer and Server to Client


Server
HTTPS
Websocket
HTTPS
Websocket
Client A
Client B


SRTP
DTLS
Client A
Client B