MFi Hardware Bypass in Wireless CarPlay Using DeviceIdentity

BAA Credentials as Receiver Authentication Material

Amine Rostane

hello@aminerostane.com

May 10, 2026

Contents

1 | Abstract

A jailbroken iPad can act as a software-only wireless CarPlay receiver without an external MFi authentication coprocessor. The receiver advertises the same Bluetooth and AirPlay surfaces that an automotive head unit exposes, completes iAP2 accessory authentication with Apple-issued BAA material, moves the iPhone onto the iPad hotspot, then completes AirPlay pairing and MFi-SAP authentication over RTSP with BAA material issued by the same local DeviceIdentity service.

The result gives the iPhone enough evidence to treat the iPad as a CarPlay receiver. The iPhone sends the H.264 screen stream over an encrypted data socket and accepts touch input as HID reports over the encrypted event channel. The bypass does not remove authentication. It replaces the expected external MFi accessory credential with a device-bound iPad credential issued by Apple's Basic Attestation Authority.

Main Finding

An iOS device with root-level control over Bluetooth, networking, and private framework access can reuse its own BAA-backed identity inside the CarPlay receiver authentication path. The iPhone accepts BAA-backed credentials during both iAP2 authentication and AirPlay auth-setup, then completes the wireless CarPlay session.

2 | Research Scope

The research covers a working end-to-end CarPlay receiver flow on owned test devices. It focuses on protocol behavior, credential placement, cryptographic derivation, and transport sequencing. It does not claim a remote compromise, a baseband compromise, a sandbox escape, or server-side compromise.

Item	Research boundary
Receiver platform	Jailbroken cellular iPad with root access and Personal Hotspot support.
Controller platform	iPhone using the standard Settings CarPlay pairing flow.
Credential source	Local iPad DeviceIdentity BAA certificate chain and signing key reference.
External hardware	No MFi authentication coprocessor and no USB CarPlay adapter.
Out of scope	Remote compromise, iCloud account compromise, carrier-side provisioning abuse, and persistent iPhone modification.

The receiver controls its Bluetooth stack in user space. It also runs an AirPlay-compatible RTSP service on the hotspot network. The iPhone performs the normal wireless CarPlay discovery path and opens a receiver session after the iPad sends the required iAP2 messages and publishes the AirPlay service records.

3 | Main Finding

Wireless CarPlay spans two authentication layers. The Bluetooth side uses iAP2 accessory authentication. The Wi-Fi side uses AirPlay pair-setup, pair-verify, and MFi-SAP auth-setup.

Automotive receivers satisfy those layers with MFi accessory material. A jailbroken iPad can instead source an Apple BAA chain and signing key from the local DeviceIdentity service.

Trust Boundary Note

The iPhone verifies Apple-issued authentication material, yet the protocol path does not tie that material to a physical MFi receiver coprocessor. A device credential from an iPad can occupy the receiver credential slot when the surrounding Bluetooth, iAP2, Wi-Fi, and AirPlay behavior matches CarPlay expectations.

Layer	Expected receiver role	iPad-backed receiver role
Bluetooth inquiry	Advertise CarPlay and iAP2 service UUIDs.	Publishes the same UUIDs and SDP records through BTstack.
iAP2 authentication	Return accessory certificate chain and sign challenge.	Returns BAA leaf and intermediate, signs with the iPad BAA key.
Wi-Fi handoff	Supply network information for wireless CarPlay.	Supplies Personal Hotspot SSID, password, channel, and port data.
AirPlay pairing	Establish encrypted control and event channels.	Runs SRP-3072, X25519, Ed25519, HKDF, and ChaCha20-Poly1305.
MFi-SAP auth-setup	Prove receiver identity on the RTSP side.	Signs the auth-setup transcript with the BAA key and returns the BAA chain.

4 | Threat Model

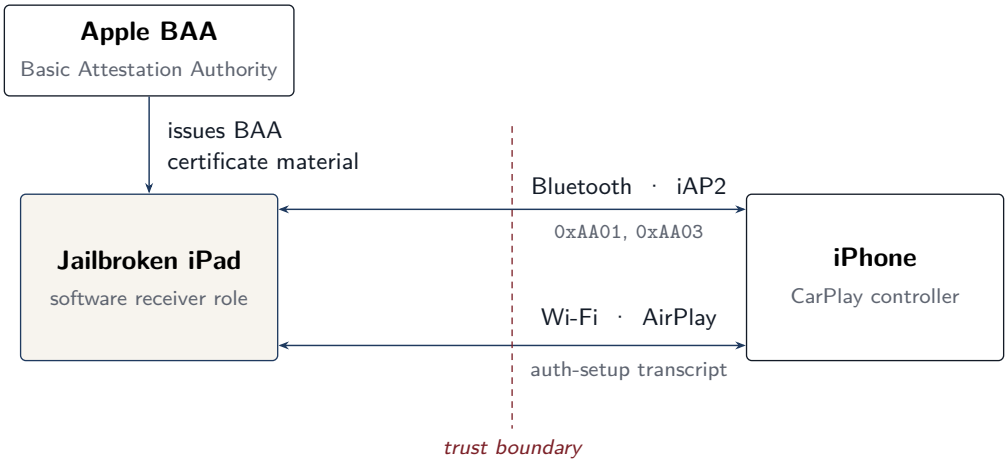
The operator has physical possession of a jailbroken cellular iPad. The operator can run unsigned code as root, unload the system Bluetooth daemon, start a user-space Bluetooth stack, query private DeviceIdentity interfaces, and bind local network ports. The iPhone user must start CarPlay pairing and choose the advertised receiver in Settings.

The operator does not need an external CarPlay dongle. The operator does not need MFi silicon. The operator does not modify the iPhone. The iPhone proceeds because the receiver reproduces the protocol surfaces and supplies Apple-issued authentication evidence.

5 | End-to-End Bypass Flow

The receiver must satisfy the iPhone in a strict order. A later AirPlay stage fails when an earlier Bluetooth or iAP2 detail deviates from the expected CarPlay profile.

```
1  Receiver takes over Bluetooth Classic inquiry and RFCOMM.
2  Receiver advertises CarPlay discovery UUID and iAP2 accessory UUID.
3  iPhone opens RFCOMM channel 3 and probes iAP2.
4  Receiver sends iAP2 SYN, receives SYN+ACK, then ACKs.
5  Receiver identifies as an iAP2 accessory with CarPlay transport support.
6  Receiver answers iAP2 authentication with BAA certificate and signature.
7  Receiver sends hotspot credentials over iAP2.
8  iPhone joins the iPad hotspot.
9  Receiver sends CarPlayStartSession with port 7000 and receiver identity.
```



Device identity material from an iPad enters the receiver authentication role.

```
10  iPhone opens AirPlay RTSP on TCP 7000.
11  Receiver completes pair-setup and pair-verify.
12  Receiver completes AirPlay auth-setup with BAA-backed MFi-SAP material.
13  Receiver answers RTSP SETUP, GET /info, and RECORD.
14  Receiver sends requestUI over the encrypted event channel.
15  iPhone sends encrypted H.264 frames and accepts HID touch reports.
```

6 | Bluetooth Discovery Surface

The iPad receiver must appear as a CarPlay-capable Classic Bluetooth accessory before the iPhone opens iAP2. The Extended Inquiry Response carries discoverability flags, two 128-bit service UUIDs, a local name, and a TX power field. The implementation uses a 240-byte EIR buffer and leaves unused bytes as zero.

Order	EIR type	Length byte	Value
1	0x01	0x02	Flags value 0x06.
2	0x07	0x21	Complete 128-bit UUID list with CarPlay discovery UUID and iAP2 accessory UUID.
3	0x09	variable	Complete local name, clamped to 30 bytes.
4	0x0A	0x02	TX power 0x00.

```
CarPlay discovery UUID
EC884348-CD41-40A2-9727-575D50BF1FD3

EIR byte order
D3 1F BF 50 5D 57 27 97 A2 40 41 CD 48 43 88 EC

iAP2 accessory UUID
00000000-DECA-FADE-DECA-DEAFDECACAFF

EIR byte order
FF CA CA DE AF DE CA DE DE FA CA DE 00 00 00 00
```

The HCI local name command receives a fixed 248-byte parameter buffer. The receiver writes the chosen name at the front and zero-fills the rest. The fixed buffer matches the Bluetooth command parameter length and avoids reading adjacent memory when the command helper copies the full block.

The receiver sets Class of Device `0x200420`. The SDP database exposes an iAP2 service on RFCOMM channel 3 and a Device ID record. The service name is `Wireless iAP`. The Device ID record uses vendor `0x02D1`, product `0x0100`, version `0x0100`, primary flag `0x01`, and source `0x0002`.

7 | iAP2 Link Establishment

The iPhone sends a six-byte detect frame before normal iAP2 packets begin. The receiver echoes the frame.

```
FF 55 02 00 EE 10
```

iAP2 frames then use a `FF 5A` preamble, a big-endian total length, a control byte, packet and acknowledgement numbers, a session id, a header checksum, payload bytes, and a payload checksum.

```
offset  size  field
0       1     0xFF
1       1     0x5A
2       2     total length, big-endian
4       1     control flags
5       1     packet sequence number
6       1     acknowledgement number
7       1     session id
8       1     header checksum
9       n     payload
9+n     1     payload checksum
```

The receiver sends this SYN payload.

```
01 05 08 00 05 DC 00 49 1E 03 01 00 02 03 02 01
```

Offset	Size	Value	Meaning
0	1	`01`	Link version.
1	1	`05`	Maximum outstanding packets.
2	2	`08 00`	Maximum received packet length.
4	2	`05 DC`	Retransmission timeout, decimal 1500.
6	2	`00 49`	Cumulative acknowledgement timeout, decimal 73.
8	1	`1E`	Maximum retransmissions, decimal 30.
9	1	`03`	Maximum cumulative acknowledgements.
10	3	`01 00 02`	Session 1, control, version 2.
13	3	`03 02 01`	Session 3, external accessory, version 1.

Session 1 carries iAP2 control messages. The inner control envelope starts with `40 40`, followed by a big-endian message length and a big-endian message id.

8 | iAP2 Identification

The iPhone sends `0x1D00`. The receiver returns `0x1D01` and declares enough capability to reach wireless CarPlay. The identification response includes name, model, manufacturer, serial, firmware revision, hardware revision, language information, an ExternalAccessoryProtocol group, a BluetoothTransportComponent group, and a WirelessCarPlayTransportComponent group.

Parameter	Type	Value
`0x0000`	UTF-8	Runtime device name.
`0x0001`	UTF-8	`RL-100`.
`0x0002`	UTF-8	`RoadLink Labs`.
`0x0003`	UTF-8	`RL100-0001`.
`0x0004`	UTF-8	`1.0`.
`0x0005`	UTF-8	`1.0`.
`0x0006`	u16 array	Outbound message id list.
`0x0007`	u16 array	Inbound message id list.
`0x0008`	u8	Zero.
`0x0009`	u16	Zero.
`0x000A`	group	ExternalAccessoryProtocol.
`0x000C`	UTF-8	`en`.
`0x000D`	UTF-8 list	`en`.
`0x0011`	group	BluetoothTransportComponent.
`0x0018`	group	WirelessCarPlayTransportComponent.

The WirelessCarPlayTransportComponent group carries component id 1, name `RoadLink WiFi`, an iAP2 support marker, and a CarPlay support marker.

Declared outbound id	Purpose
`0xAA01`	Authentication certificate.
`0xAA03`	Authentication challenge response.
`0x4E01`	Bluetooth component information.
`0x5703`	Accessory Wi-Fi configuration information.
`0xEA03`	External accessory protocol session status.

Declared device-to-receiver id	Purpose
`0xAA00`	Request authentication certificate.
`0xAA02`	Request authentication challenge.
`0xAA04`	Authentication failed.

Declared device-to-receiver id	Purpose
`0xAA05`	Authentication succeeded.
`0x5702`	Request accessory Wi-Fi configuration information.
`0x4E0B`	Device time update.
`0x4E0D`	Wireless CarPlay update.
`0x4E0E`	Device transport identifier notification.
`0xEA00`	Start external accessory protocol session.
`0xEA01`	Stop external accessory protocol session.

The implementation also handles the identification control messages `0x1D00`, `0x1D02`, and `0x1D03`. Those messages are part of the identification exchange itself, not entries in the declared post-identification receive list above. The receiver later sends `0x4301` even though the identification message does not list it in the outbound id array. In testing, the iPhone accepted the message after Wi-Fi join and transport identifier notification.

9 | BAA Use in iAP2 Authentication

The Bluetooth authentication exchange uses iAP2 messages `0xAA00` through `0xAA05`. The iPhone first asks for a certificate. The receiver returns a BAA leaf certificate, an authentication type byte, and a BAA intermediate certificate. The iPhone then sends a challenge. The receiver signs the challenge with ECDSA-SHA256 using the private key associated with the BAA leaf.

Message	Parameter	Type	Meaning
`0xAA01`	`0x0000`	blob	BAA leaf DER.
`0xAA01`	`0x0001`	u8	Authentication type `0x01`.
`0xAA01`	`0x0002`	blob	BAA intermediate DER.
`0xAA03`	`0x0000`	blob	ECDSA-SHA256 signature over the iPhone challenge.

The certificate response uses native iAP2 parameters. It does not wrap the chain in PKCS#7. The signature response contains the raw DER ECDSA signature blob expected by the iAP2 challenge path.

Authentication Pivot

The iPhone asks for accessory authentication during iAP2 setup. The receiver supplies an iPad BAA chain and challenge signature in the accessory response fields. The iPhone then sends `0xAA05`, allowing the rest of the CarPlay bootstrap to continue.

10 | Wi-Fi Handoff

The iPhone sends `0x5702` after it accepts the receiver identity. The receiver answers `0x5703` with hotspot credentials.

Parameter	Type	Value
`0x0001`	UTF-8 with NUL	Personal Hotspot SSID.
`0x0002`	UTF-8 with NUL	Personal Hotspot password.
`0x0003`	u8	WPA or WPA2 security marker.
`0x0004`	u8	Wi-Fi channel.

The receiver omits BSSID parameter `0x0005`. The iPhone joins the hotspot and later sends `0x4E0E` with transport identifiers, including its Wi-Fi MAC and USB serial. The receiver then sends `0x4301`, the CarPlay start-session message.

Parameter	Type	Value
`0x0001`	group	WirelessAttributes.
`0x0002`	u32	AirPlay RTSP port 7000.
`0x0003`	UTF-8 with NUL	Receiver device id `90:B9:31:AC:86:A0`.
`0x0004`	UTF-8 with NUL	Receiver Ed25519 public key as lowercase hex.
`0x0005`	UTF-8 with NUL	Source version `280.33.8`.

WirelessAttributes item	Type	Value
`0x0000`	UTF-8 with NUL	Hotspot SSID.
`0x0001`	UTF-8 with NUL	Hotspot password.
`0x0002`	u8	Channel.
`0x0003`	UTF-8 with NUL	Link-local IPv6 address on the hotspot bridge when available.

11 | AirPlay Discovery

The receiver publishes `_airplay._tcp` and `_raop._tcp` on TCP port 7000 once the hotspot bridge is available. The iPhone observes those Bonjour records after Wi-Fi handoff, then uses them to locate the RTSP endpoint and verify that the target advertises CarPlay-compatible AirPlay features.

Service	Key	Value
`_airplay._tcp`	`deviceid`	`90:B9:31:AC:86:A0`
`_airplay._tcp`	`features`	`0x44540380,0x61`
`_airplay._tcp`	`flags`	`0x4`
`_airplay._tcp`	`model`	`AirPlayGeneric1,1`
`_airplay._tcp`	`protovers`	`1.1`
`_airplay._tcp`	`pi`	`29f0a5dc-2c2a-4b3e-9e5d-1a6c85f201a3`
`_airplay._tcp`	`pk`	Receiver Ed25519 public key
`_airplay._tcp`	`srcvers`	`509.0`
`_raop._tcp`	`am`	`AirPlayGeneric1,1`
`_raop._tcp`	`ft`	`0x44540380,0x61`
`_raop._tcp`	`pk`	Receiver Ed25519 public key
`_raop._tcp`	`vs`	`509.0`

The RAOP record also advertises audio transport compatibility through `txtvers`, `ch`, `cn`, `da`, `et`, `md`, `pw`, `sv`, `sr`, `ss`, `tp`, `vn`, and `sf`. The receiver focuses on screen and touch transport, while the advertised audio fields keep the AirPlay profile close to receiver expectations.

12 | AirPlay Pair-Setup

The iPhone opens RTSP on TCP 7000 and sends TLV8 messages to `POST /pair-setup`. The receiver implements the SRP-6a flow used by AirPlay and HomeKit.

Property	Value
Group	3072-bit MODP group from RFC 3526 and RFC 5054.
Generator	5.
Hash	SHA-512.
Username	`Pair-Setup`.
PIN	`3939`.
Salt length	16 bytes.
Session key	`SHA512(S)` with one hash operation.

$$
\begin{aligned}
x &= H(s \parallel H(I \parallel : \parallel p)) \\
v &= g^x \bmod N \\
k &= H(\mathrm{PAD}(N) \parallel \mathrm{PAD}(g)) \\
B &= kv + g^b \bmod N \\
u &= H(\mathrm{PAD}(A) \parallel \mathrm{PAD}(B)) \\
S &= (A \cdot v^u)^b \bmod N \\
K &= H(S)
\end{aligned}
$$

The iPhone proof uses this construction.

$$
\begin{aligned}
M_1 = H(H(N) \oplus H(g) \parallel H(I) \parallel s \parallel \\
\mathrm{PAD}(A) \parallel \mathrm{PAD}(B) \parallel K)
\end{aligned}
$$

The receiver proof uses `H(PAD(A) || M1 || K)`.

Step	Direction	State	Payload
M1	iPhone to receiver	1	Method `PairSetup`.
M2	Receiver to iPhone	2	Salt and SRP public key `B`.
M3	iPhone to receiver	3	SRP public key `A` and proof `M1`.
M4	Receiver to iPhone	4	Server proof `M2`.
M5	iPhone to receiver	5	Encrypted controller identity sub-TLV.

Step	Direction	State	Payload
M6	Receiver to iPhone	6	Encrypted receiver identity sub-TLV.

M5 and M6 use ChaCha20-Poly1305 with the pair-setup encryption key.

```
salt = Pair-Setup-Encrypt-Salt
info = Pair-Setup-Encrypt-Info
M5 decrypt nonce = PS-Msg05
M6 encrypt nonce = PS-Msg06
```

The receiver signs its M6 identity sub-TLV with Ed25519. The signature input combines the accessory signing context, the receiver identifier, and the receiver public key.

```
accessoryX = HKDF(K,
                  Pair-Setup-Accessory-Sign-Salt,
                  Pair-Setup-Accessory-Sign-Info)[0..31]
signature = Ed25519(accessoryX || receiver_identifier || receiver_public_key)
```

The receiver stores the controller Ed25519 public key from M5. The tested implementation parses and logs the M5 signature field, but it does not verify that signature during pair-setup. Controller signature verification happens later during pair-verify, and the tested implementation logs a warning rather than aborting when that later verification fails.

13 | AirPlay Pair-Verify

The iPhone sends `POST /pair-verify` after pair-setup. The receiver generates a fresh X25519 keypair and derives a 32-byte encryption key from the shared secret.

```
salt = Pair-Verify-Encrypt-Salt
info = Pair-Verify-Encrypt-Info
M2 encrypt nonce = PV-Msg02
M3 decrypt nonce = PV-Msg03
```

Step	Direction	State	Payload
M1	iPhone to receiver	1	Controller X25519 public key.
M2	Receiver to iPhone	2	Receiver X25519 public key and encrypted receiver identity.
M3	iPhone to receiver	3	Encrypted controller identity.
M4	Receiver to iPhone	4	Success state.

The M2 receiver signature covers these byte strings.

```
receiver_x25519_public || receiver_identifier || controller_x25519_public
```

The M3 controller signature covers the mirrored order.

```
controller_x25519_public || controller_identifier || receiver_x25519_public
```

The receiver attempts Ed25519 verification when it has the controller public key from pair-setup. The tested implementation logs a verification warning and still sends M4 when verification fails or when the key is missing.

14 | Encrypted RTSP Channels

Pair-verify gives both sides the shared secret used for RTSP control and event channel keys. The labels use the controller point of view. The receiver therefore reads with the controller write key and writes with the controller read key on the RTSP control channel.

Channel	Receiver use	HKDF labels
Control	read	`Control-Salt` and `Control-Write-Encryption-Key`.
Control	write	`Control-Salt` and `Control-Read-Encryption-Key`.
Event	read	`Events-Salt` and `Events-Read-Encryption-Key`.
Event	write	`Events-Salt` and `Events-Write-Encryption-Key`.

Encrypted RTSP records use a two-byte little-endian plaintext length as additional authenticated data. The nonce stores four zero bytes followed by an eight-byte little-endian counter.

```
record = plaintext_len_le16 || ciphertext || poly1305_tag
aad    = plaintext_len_le16
nonce  = 00 00 00 00 || counter_le64
```

15 | BAA Use in AirPlay Auth-Setup

The iPhone sends `POST /auth-setup` after pair-verify. The receiver accepts a raw 33-byte MFi-SAP v1 body and a binary-plist wrapped variant. The raw form contains one version byte followed by a 32-byte X25519 controller public key.

The receiver creates a fresh X25519 keypair and derives AES-128-CTR material from the shared secret with SHA-1.

```
aesKeyMaterial = SHA1("AES-KEY" || sharedSecret)
aesIVMaterial  = SHA1("AES-IV"  || sharedSecret)
AES key        = first 16 bytes of aesKeyMaterial
AES IV         = first 16 bytes of aesIVMaterial, CTR-BE mode
```

The receiver signs this 64-byte transcript with ECDSA-SHA256 using the BAA private key.

```
receiver_x25519_public || controller_x25519_public
```

The response body sends the receiver X25519 public key, the BAA leaf certificate, the AES-CTR encrypted ECDSA signature, and an OPACK dictionary whose `baIC` value contains the BAA intermediate certificate.

```
server_x25519_public[32]
leaf_len_be32
leaf_der[leaf_len]
encrypted_signature_len_be32
encrypted_signature[encrypted_signature_len]
```

```
baIC_opack_len_be32
OPACK({"baIC": intermediate_der})[baIC_opack_len]
```

Second Credential Acceptance Point

The iPhone verifies BAA-backed MFi-SAP material on the AirPlay RTSP side after it already accepted BAA-backed iAP2 authentication on Bluetooth. Both checks allow the software receiver to continue without MFi accessory silicon.

16 | RTSP and CarPlay Session Start

After pair-verify and auth-setup, the iPhone drives the RTSP session. The receiver accepts the control connection on an IPv6 socket with IPv4-mapped addresses allowed.

Order	Request	Receiver action
1	`POST /pair-setup`	Runs SRP pair-setup M1 through M6.
2	`POST /pair-verify`	Runs X25519 pair-verify M1 through M4 and derives RTSP keys.
3	`POST /auth-setup`	Runs BAA-backed MFi-SAP authentication.
4	`SETUP`	Creates timing, event, and keepalive ports.
5	Event TCP connect	Accepts encrypted event-channel connection.
6	`GET /info`	Sends receiver capabilities as a binary plist.
7	`RECORD`	Starts timing, event, and keepalive threads.
8	Timing negotiation	Sends five NTP-style timing requests and waits for at least three responses.
9	Event `requestUI`	Sends `\{"type":"requestUI"\}` on the encrypted event channel.
10	Stream `SETUP`	Creates a data socket for the screen stream.
11	Screen TCP connect	Accepts encrypted H.264 frame stream.

The initial `SETUP` response returns a binary plist with `timingPort`, `eventPort`, and `keepAlivePort`. The receiver records the iPhone timing port from the request and uses the peer address from the control socket.

17 | Receiver Capability Plist

`GET /info` returns a binary plist that declares the media and input surface. The device identifiers and AirPlay feature flags must match the Bonjour records and `0x4301` values.

Key	Type	Value
`deviceID`	string	`90:B9:31:AC:86:A0.`
`macAddress`	string	`90:B9:31:AC:86:A0.`

Key	Type	Value
`features`	integer	`0x6144540380` in the current HomeKit-pairing-enabled build.
`model`	string	`AirPlayGeneric1,1`.
`name`	string	Runtime receiver name.
`manufacturer`	string	`iPadPlay`.
`sourceVersion`	string	`509.0`.
`protocolVersion`	string	`1.1`.
`statusFlags`	integer	`0x4`.
`pi`	string	Receiver pairing UUID.
`pk`	data	Receiver Ed25519 public key.
`modes`	dictionary	Resource requests for MainScreen and MainAudio.
`displays`	array	Single 800 by 480 display at 30 FPS.
`hidDevices`	array	Single touchscreen HID device.
`audioFormats`	array	Stream type 100 mask `0x01000000`; stream type 96 mask `0x04000000`.

The display entry uses UUID `e0ff8a27-6738-3d56-8a16-cc53ce1299b4`, width 800, height 480, maximum frame rate 30, feature mask `0x1A`, and primary input device 1. The resource map requests MainScreen and MainAudio with transfer type 1, priority 500, take constraint 100, and borrow constraint 100.

18 | Timing Negotiation

The receiver runs a UDP timing responder and also initiates timing negotiation before it answers `RECORD`. It sends five 32-byte RTCP-style timing request packets to the iPhone timing port with 100 ms spacing. It waits up to 2000 ms for at least three responses.

```
request byte 0 = 0x80
request byte 1 = 0xD2
request byte 2 = 0x00
request byte 3 = 0x07
transmit time = NTP seconds and fraction at bytes 24..31
```

The receiver answers `RECORD` with `200 OK` after the timing negotiation attempt finishes. It records whether at least three timing responses arrived, but the current implementation does not reject `RECORD` when fewer responses arrive before the wait path ends. The iPhone then proceeds to the event-channel `requestUI` trigger and screen stream setup.

19 | Encrypted Screen Stream

The second RTSP `SETUP` creates a screen data socket. The iPhone supplies a stream connection id. The receiver derives the stream key with HKDF-SHA512.

```
salt = "DataStream-Salt" || decimal(streamConnectionID)
info = "DataStream-Output-Encryption-Key"
key  = HKDF-SHA512(pairVerifySharedSecret, salt, info)[0..31]
```

Each screen message starts with a 128-byte header. The first four bytes encode body size as little-endian u32. Byte 4 stores the opcode.

```
offset  size  field
0       4     bodySize, little-endian
4       1     opcode
5       3     smallParam
8       120   fifteen Value64 parameters
```

Opcode	Name	Handling
`0x00`	VideoFrame	Decrypt body and forward AVCC H.264 data.
`0x01`	VideoConfig	Treat body as plaintext AVCC SPS and PPS data.
`0x02`	KeepAlive	Ignore.
`0x04`	Ignore	Ignore bandwidth-measurement body.
`0x05`	KeepAliveWithBody	Ignore encoder statistics body.

For VideoConfig, the receiver reads width from `header + 16` and height from `header + 20`. Those offsets correspond to `params[1].f32[0]` and `params[1].f32[1]`.

VideoFrame bodies use ChaCha20-Poly1305. The additional authenticated data covers the entire 128-byte header. The nonce uses four zero bytes followed by an eight-byte little-endian frame counter. The body stores ciphertext followed by a 16-byte Poly1305 tag.

```
aad        = header[0..127]
nonce      = 00 00 00 00 || frameCounter_le64
ciphertext = body[0 .. bodySize - 17]
tag        = body[bodySize - 16 .. bodySize - 1]
```

20 | Touch Return Channel

The receiver declares one touchscreen HID device in the `GET /info` plist. The descriptor uses Digitizer usage page `0x0D`, Touch Screen usage `0x04`, a Finger logical collection, one one-bit touch field, seven padding bits, and two 16-bit absolute axes.

```
05 0D 09 04 A1 01 05 0D 09 22 A1 02 05 0D 09 33
15 00 25 01 75 01 95 01 81 02 75 07 95 01 81 01
05 01 09 30 15 00 26 20 03 75 10 95 01 81 02 09
31 15 00 26 E0 01 75 10 95 01 81 02 C0 C0
```

The logical X range is 0 through 800. The logical Y range is 0 through 480. A HID report uses five bytes.

```
byte 0      touch bit
bytes 1..2  x, little-endian
bytes 3..4  y, little-endian
```

The receiver sends touch input over the encrypted event channel as an RTSP `POST /command` whose body is a binary plist command.

```
{
  "type"      = "hidSendReport",
```

```
  "uuid"      = "1",
  "hidReport" = five_byte_report
}
```

21 | Empirical Constraints

The tested receiver depends on the iPad Personal Hotspot stack for AP mode, DHCP, WPA2, IPv6 link-local behavior, and NAT. It does not implement those networking functions itself.

Testing found that the iPhone refused to auto-join some Personal Hotspot names. Receiver-side validation therefore requires at least six SSID characters and rejects lowercase matches for `ipad`, `iphone`, and `ipod`. Treat this as an empirical compatibility rule from the tested devices rather than an Apple-documented requirement.

The receiver uses a static Ed25519 receiver identity. It does not persist trusted controller records across runs. It advertises audio resources and binds related ports, while the working receiver path centers on screen mirroring and touch return.

22 | Security Impact

The finding reduces the hardware requirement for a wireless CarPlay receiver. A jailbroken iPad can satisfy the iPhone's receiver checks without an MFi coprocessor by placing a BAA-backed iPad identity into the accessory authentication and AirPlay auth-setup paths.

The impact stays local and interactive under the tested conditions. The iPhone user still selects the receiver during CarPlay setup. The receiver must control Bluetooth discovery, complete iAP2, host the Wi-Fi network, and answer AirPlay pairing. The bypass affects device classification and receiver trust, not iPhone code execution.

Impact area	Result
MFi hardware gate	External authentication silicon no longer gates the proof-of-concept receiver.
Receiver role substitution	A general iPad can present as a CarPlay receiver when jailbroken and configured.
Credential reuse	BAA material issued for the iPad functions as receiver authentication evidence.
Session media and input access	The iPhone sends screen frames and accepts touch input after pairing and auth-setup finish.
Remote applicability	The tested path needs local radio proximity, user pairing action, and a controlled receiver device.

23 | Protocol Constants

Name	Value
CarPlay discovery UUID	`EC884348-CD41-40A2-9727-575D50BF1FD3`.
iAP2 accessory UUID	`00000000-DECA-FADE-DECA-DEAFDECACAFF`.

Name	Value
iAP2 RFCOMM channel	3.
iAP2 detect frame	`FF 55 02 00 EE 10`.
iAP2 control session	1.
iAP2 external accessory session	3.
AirPlay TCP port	7000.
Receiver device id	`90:B9:31:AC:86:A0`.
RAOP device id	`90B931AC86A0`.
AirPlay model	`AirPlayGeneric1,1`.
AirPlay source version	`509.0`.
`0x4301` source version	`280.33.8`.
Pairing UUID	`29f0a5dc-2c2a-4b3e-9e5d-1a6c85f201a3`.
Ed25519 public key	`1b15f0ad 62c89472 1c409765 1801e628 45451a18 3c8df8af 7d6b2043 0823586f`.
Screen resolution	800 by 480.
Screen frame rate	30 FPS.
HID report length	5 bytes.

24 | Conclusion

The receiver succeeds because it combines four conditions in one controlled device. It presents the right Bluetooth discovery and iAP2 identity, it signs iAP2 authentication with BAA material, it moves the iPhone onto a usable Wi-Fi path, and it completes AirPlay pairing plus auth-setup with the same device-backed credential family. Once those checks pass, the iPhone behaves as it does with a wireless CarPlay head unit. It sends encrypted video frames and accepts HID touch reports.

The core security lesson concerns credential purpose. A valid Apple-issued iPad identity can authenticate a CarPlay receiver role when iOS accepts it inside the receiver protocol flow. The protocol still uses strong cryptography. The bypass comes from where the credential enters the trust model.