The Web Authentication Guide Cheatsheet

	Complexity	Reliance on HTTPS	Protection against			Use-cases	Taking care
			CSRF	Replay	Tampering		
HTTP Basic	Easy	Full	No	No	No	• No session management • Read-only resource • Backend compatibility	• Hash secrets on server side • Use with TLS
HTTP Digest	Medium	Confidentiality	No	Yes	Yes[1]	• Need something stronger than Basic w/ high compatibility	• Use full featured libraries • Use a unique realm
Cookies	Easy	Full	No[2]	No	No	• Full blown session management	• Set the correct flags: Secure, HTTP Only, SameSite • Add prefixes for added security[3]
Bearer Tokens	Easy	Full[4]	Yes	No	No[4]	• Limited session management needs • OAuth and other backend integrations	• Use OAuth for integrations • Keep the JWT secret safe • Utilize short-lived tokens • Consider using access and refresh tokens
Signature Schemes	Hard	Confidentiality	Yes	Yes	Yes	• Use on the backend between servers • Use to provide pre-signed URLs	• Choose your library carefully • Keep secrets safe
TLS Client Certificates	Hard	Full[5]	No	Yes	Yes	• Elevated security requirements • Use on the backend between servers	• Harden TLS settings • Use strong keys[6]

1. Only the request method and URI by default. Modern implementations supporting the "digest" property can protect the request body as well.
2. SameSite attribute turns this into a Yes.
3. __Secure- and __Host-
4. JWT provides integrity protection for the token itself.
5. This makes it impossible for an active network attacker to spy on the connection.
6. At least 2048 bits for RSA and 256 bits for ECC.