Tecnologie e applicazioni web

Authentication

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia

Academic year: 2023/2024

Authentication

Cookies allow a web server to store key-value pairs to the user agent requesting a particular resource

With this technique, we can recognize the same client throughout subsequent requests but not its “real identity”

HTTP supports various mechanisms to **authenticate** a client by providing its credentials

Authentication

Authentication means **showing some evidence of the actual physical identity of a particular client**

It is usually based on some **shared information** between client and server (ex. username-password pair) that must be exchanged securely with a pre-defined protocol

HTTP Authentication

When a client requests a protected resource, the server may respond with the status code **`401`** (login required)

Together with the status code, the **`WWW-authenticate`** header informs the user agent to the kind of data that must be provided to authenticate

User login

The web browser, according to the kind of authentication requested:

- Ask the user for a user/password pair
- Creates an HTTP header containing the login credentials
- Uses the header for all the subsequent resources under a specific "authorization realm"

Similar to the cookie mechanism: authentication credentials are exchanged in HTTP **headers**.

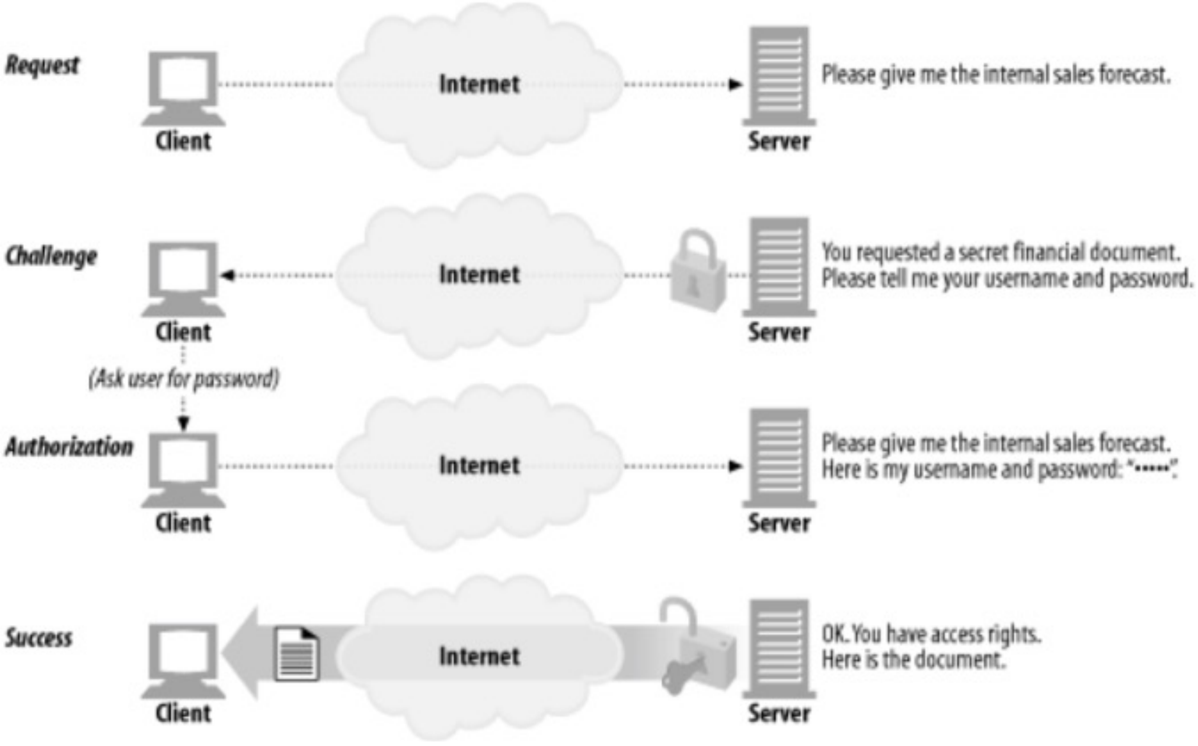
Authentication types

HTTP implements 2 authentication mechanisms:

1. Basic access authentication
2. Digest access authentication

Both are based on a challenge-response framework but differ in how the information is encoded and exchanged.

Challenge-response


Request
Client
Internet
Server
Please give me the internal sales forecast.
Challenge
Client
Internet
Server
You requested a secret financial document.
Please tell me your username and password.
(Ask user for password)
Authorization
Client
Internet
Server
Please give me the internal sales forecast.
Here is my username and password: "•••••".
Success
Client
Internet
Server
OK. You have access rights.
Here is the document.

< Challenge

< Response
(sent in the request following the challenge)

Basic authentication

Initially described in HTTP/1.0, nowadays implemented in all web browsers.

- A server may reject a transaction, challenging the client to send a `username:password` pair.
- If the reply is correct, the resource is sent in the next transaction
- If the reply is not correct, the resource is not sent, and the challenge is repeated

Basic authentication

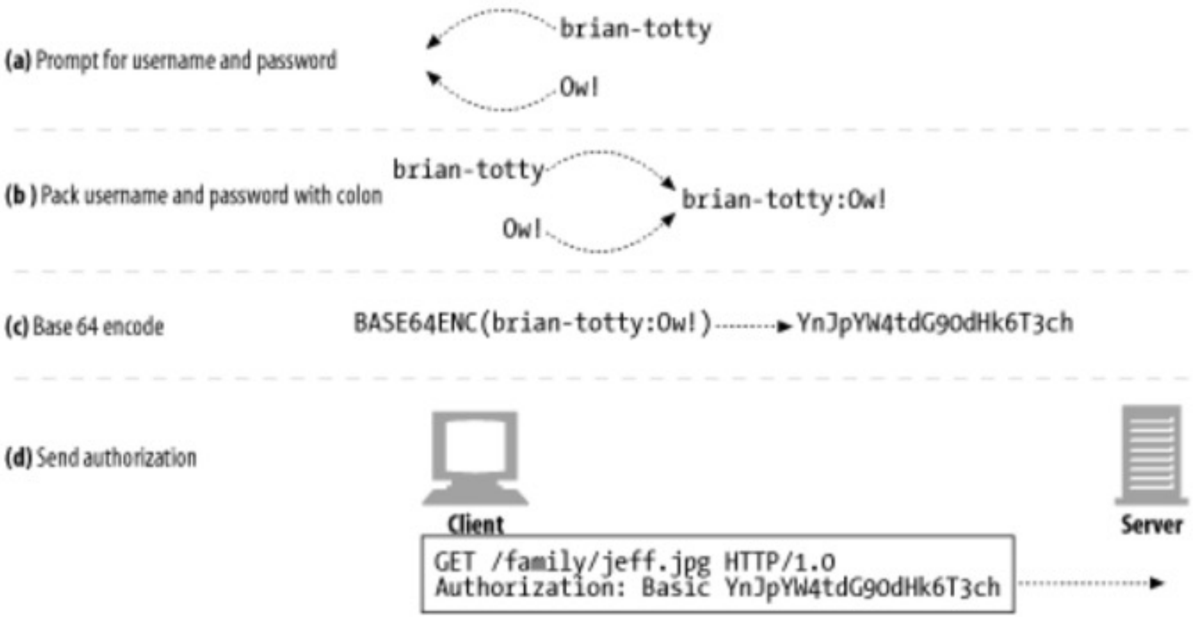
1. The client sends a **request** (GET, POST, HEAD, etc.) to access a certain resource
2. If the resource is protected, the server inserts the following header in the **response**:

```
WWW-Authenticate: Basic realm=”<realm-name>”
```

3. The client asks username and password to the user and encodes the data as a string:

<crd> = `base64(<username>:<password>)`

Base-64 encoding



```
GET /family/jeff.jpg HTTP/1.0
Authorization: Basic YnJpYW4tdG9OdHk6T3ch
```

Basic authentication

4. The user agent, for every subsequent request, inserts the header

   ```
   Authorization: Basic <crd>
   ```

The server decodes <crd> in base64 to obtain the <username>:<password> string and verifies the credentials. If valid, the resource is provided as it happens without authentication.

Base-64 encoding

Base64 encodes a generic byte stream to a string containing alphanumeric characters only.

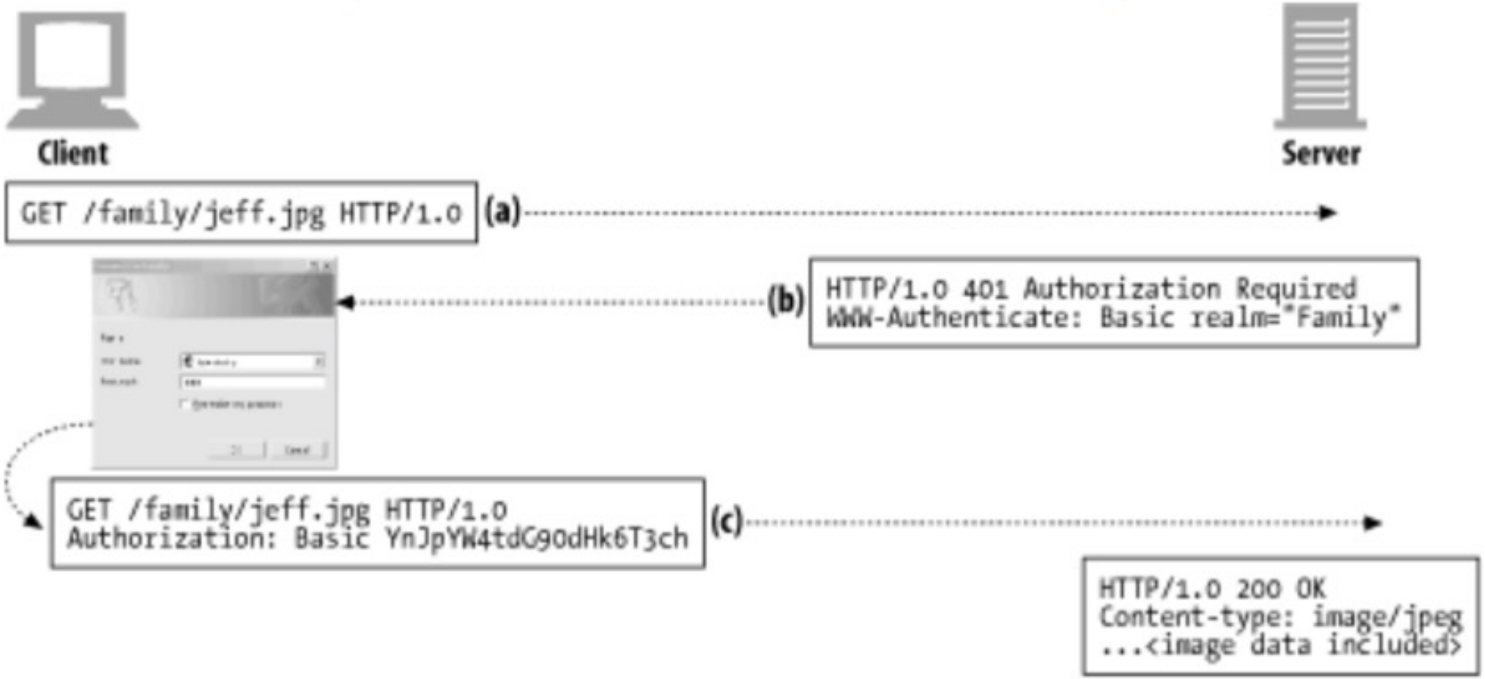
Important: Base64 is not meant to encrypt the data that remains easily readable to anyone. The user:password pair is just obfuscated (not secure at all)

Base-64 encoding

Why not just send <username>:<password> without encoding?

1. Non plain-ascii characters in the password can be safely inserted into the HTTP headers
2. String is obfuscated to prevent humans from reading the password if the HTTP traffic is observed

Basic authentication



```
GET /family/jeff.jpg HTTP/1.0
```

```
HTTP/1.0 401 Authorization Required
WWW-Authenticate: Basic realm="Family"
```

```
GET /family/jeff.jpg HTTP/1.0
Authorization: Basic YnJpYW4tdG90dHk6T3ch
```

```
HTTP/1.0 200 OK
Content-type: image/jpeg
...<image data included>
```

BA: problems

1. Anyone can easily decode user credentials. This is the same security level as sending user/password without encryption.
2. Even if the authentication is used for non-critical applications, users may still recycle passwords used for other websites or sensible applications

Solution: Use basic authentication with HTTPS only

BA: problems

3. Even if the authentication happens correctly, the provided resource is not necessarily bound to the provided credentials.

A man-in-the-middle can change the resource data without tampering the authentication headers

Solution: Again, basic authentication should only be used with HTTPS

BA: problems

4. **Server spoofing**: client cannot verify the server's true identity that can therefore be impersonated by a malicious entity.

Solution: Use digital certificates to authenticate the server before using basic access authentication (again, HTTPS solves this problem)

Digest Authentication

Basic Authentication is insecure because the username/password pair is sent without encryption.

But.. symmetric/asymmetric key techniques to create a secure channel tend to overcomplicate the protocol

Digest Authentication

Rationale:

A server does not necessarily need to receive the password (shared secret) from the client. A **digest** is sufficient to prove that a client knows the correct password

Hashing functions are commonly used to compute those digest

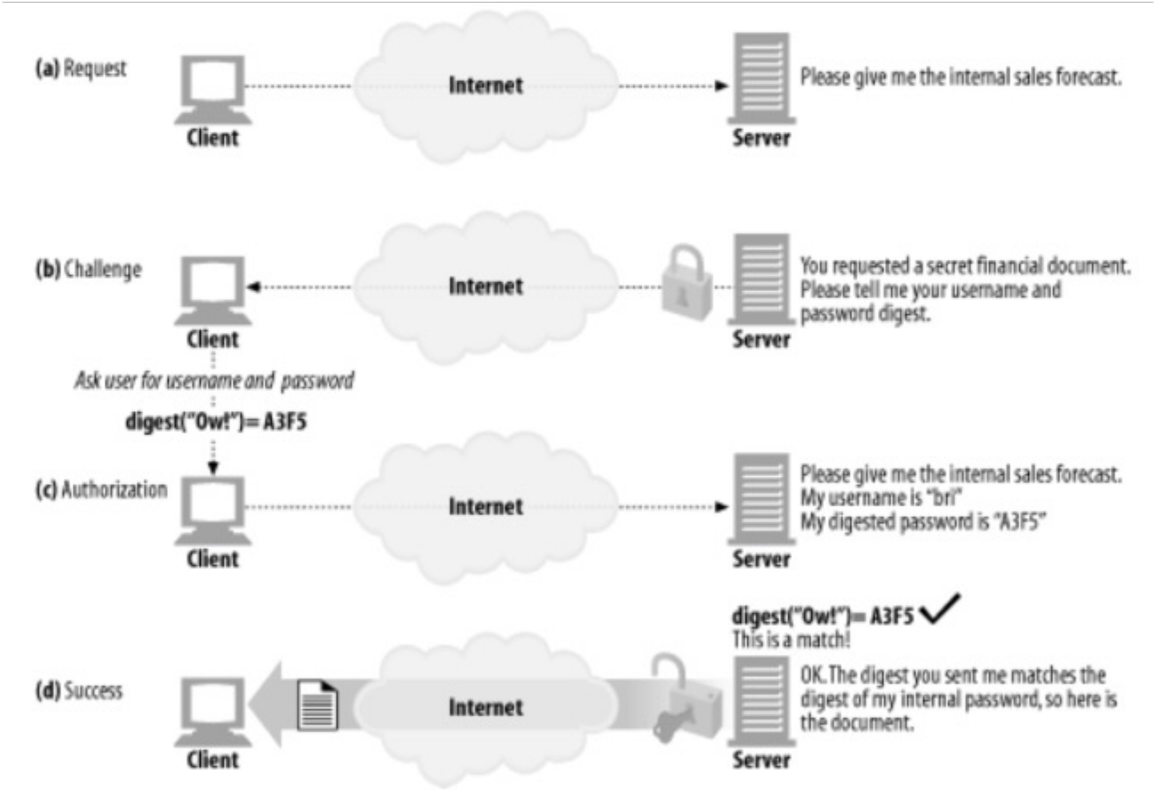
Hash function

A hash function can convert any message (string) to a fixed-length sequence of (random-like) bytes

Features:

- Same messages generate same hashes
- **A hash function is one-way**: practically impossible to recover the original message from its hash
- Collisions are possible but extremely rare

Digest authentication


(a) Request
Client
Internet
Server
Please give me the internal sales forecast.
(b) Challenge
Client
Internet
Server
You requested a secret financial document. Please tell me your username and password digest.
Ask user for username and password
digest("Ow!")= A3F5
(c) Authorization
Client
Internet
Server
Please give me the internal sales forecast.
My username is "bri"
My digested password is "A3F5"
digest("Ow!")= A3F5 ✓
This is a match!
(d) Success
Client
Internet
Server
OK. The digest you sent me matches the digest of my internal password, so here is the document.

Digest authentication

With a hash function we can avoid sending the password directly!

Problem: Since a password always generates the same digest, anyone eavesdropping on the channel can steal the digest and use it to authenticate without knowing the password: **Replay attack**

Nonce

To avoid the replay-attack, the server sends a special token (called **nonce**) as part of the authentication challenge

The client computes hash(<password>,<nonce>) so that each digest is different and usable only once (supposing that nonces change every time)

Digest Authentication

The (simplified) digest authentication protocol works as follows:

1. Similar to BA, the server sends the `WWW-Authenticate` header in its response, specifying the authentication realm and a randomly generated nonce

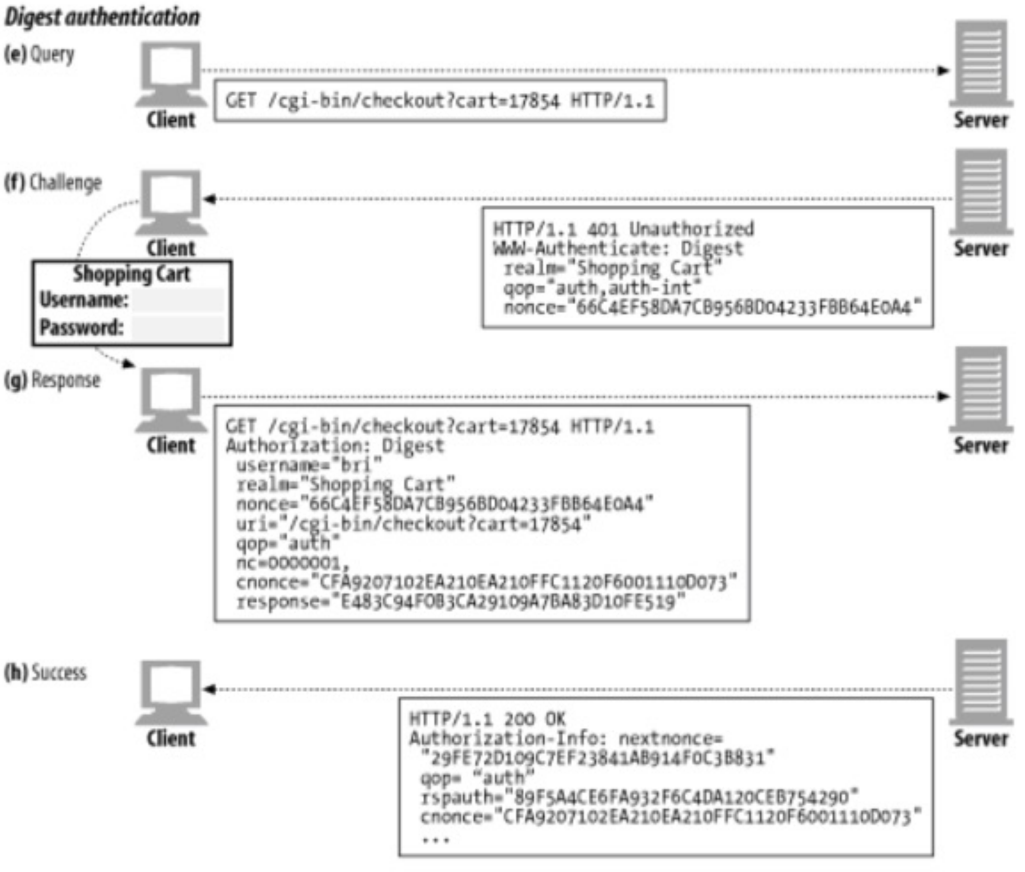
Digest Authentication

2. The client creates the digest of the triple (user, nonce, password)
3. The client sends in the Authorization header its username and the digest.
4. The server creates its digest (with user, nonce, and password) to check if it matches the one sent by the client

Digest Authentication

5. If the two digests match, the client is authenticated. If not, the request is refused and a new challenge (ie. a new nonce) is generated
6. If the client had, in turn, sent a nonce, the digest for the client is generated and returned in the Authorization-Info header

Digest Authentication



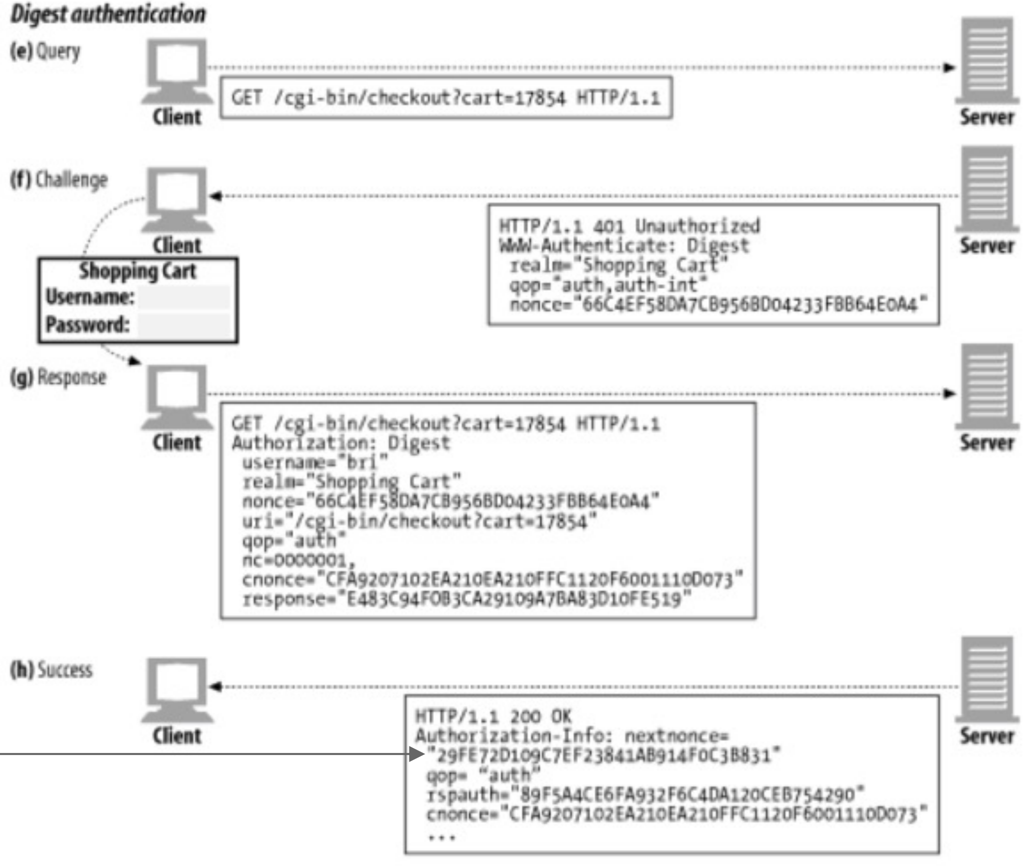
```
GET /cgi-bin/checkout?cart=17854 HTTP/1.1
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
 realm="Shopping Cart"
 qop="auth,auth-int"
 nonce="66C4EF58DA7CB956BD04233FBB64E0A4"
```

```
GET /cgi-bin/checkout?cart=17854 HTTP/1.1
Authorization: Digest
 username="bri"
 realm="Shopping Cart"
 nonce="66C4EF58DA7CB956BD04233FBB64E0A4"
 uri="/cgi-bin/checkout?cart=17854"
 qop="auth"
 nc=0000001,
 cnonce="CFA9207102EA210EA210FFC1120F6001110D073"
 response="E483C94F0B3CA29109A7BA83D10FE519"
```

```
HTTP/1.1 200 OK
Authorization-Info: nextnonce=
 "29FE72D109C7EF23841AB914F0C3B831"
 qop= "auth"
 rspauth="89F5A4CE6FA932F6C4DA120CEB754290"
 cnonce="CFA9207102EA210EA210FFC1120F6001110D073"
 ...
```

Digest Authentication



```
GET /cgi-bin/checkout?cart=17854 HTTP/1.1
```

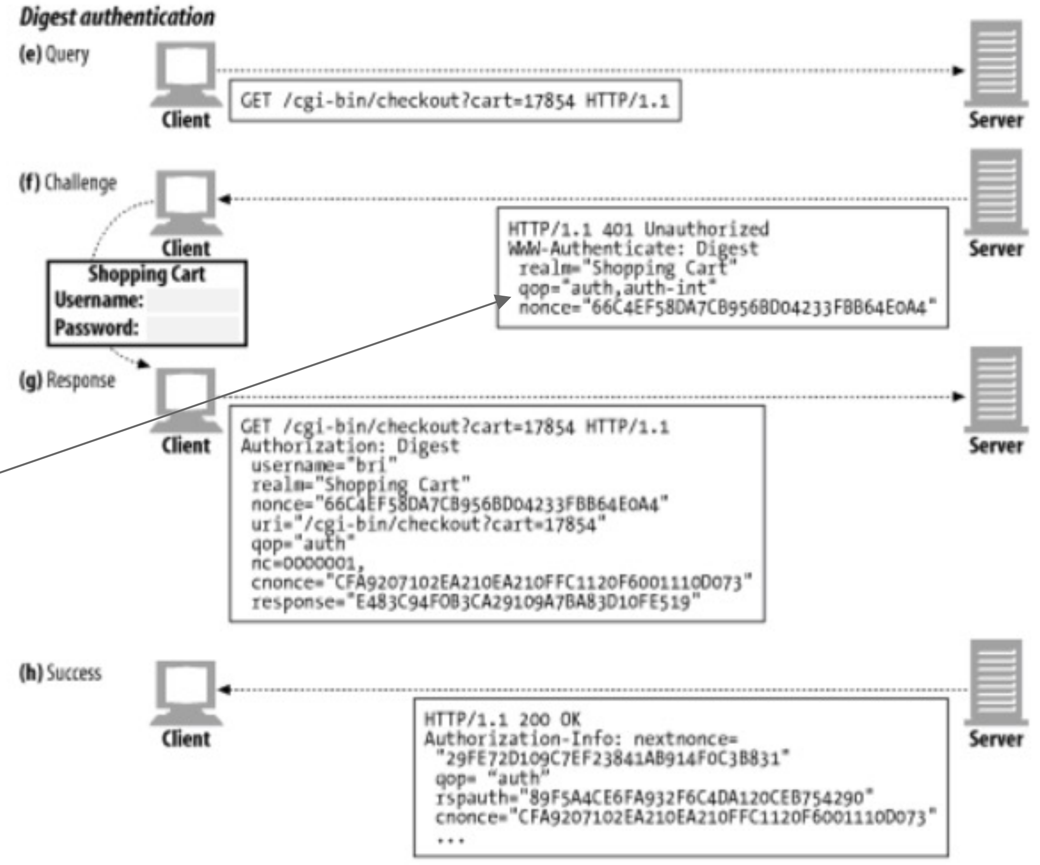
```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
 realm="Shopping Cart"
 qop="auth,auth-int"
 nonce="66C4EF58DA7CB956BD04233FBB64E0A4"
```

```
GET /cgi-bin/checkout?cart=17854 HTTP/1.1
Authorization: Digest
 username="bri"
 realm="Shopping Cart"
 nonce="66C4EF58DA7CB956BD04233FBB64E0A4"
 uri="/cgi-bin/checkout?cart=17854"
 qop="auth"
 nc=0000001,
 cnonce="CFA9207102EA210EA210FFC1120F6001110D073"
 response="E483C94FOB3CA29109A7BA83D10FE519"
```

```
HTTP/1.1 200 OK
Authorization-Info: nextnonce=
 "29FE72D109C7EF23841AB914F0C3B831"
 qop= "auth"
 rspauth="89F5A4CE6FA932F6C4DA120CEB754290"
 cnonce="CFA9207102EA210EA210FFC1120F6001110D073"
 ...
```

nextnonce allows the client to send a new request with the Authorization header already present, skipping a new challenge initiated by the server

Digest Authentication



```
GET /cgi-bin/checkout?cart=17854 HTTP/1.1
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
 realm="Shopping Cart"
 qop="auth,auth-int"
 nonce="66C4EF58DA7CB956BD04233FBB64E0A4"
```

```
GET /cgi-bin/checkout?cart=17854 HTTP/1.1
Authorization: Digest
 username="bri"
 realm="Shopping Cart"
 nonce="66C4EF58DA7CB956BD04233FBB64E0A4"
 uri="/cgi-bin/checkout?cart=17854"
 qop="auth"
 nc=0000001,
 cnonce="CFA9207102EA210EA210FFC1120F6001110D073"
 response="E483C94F0B3CA29109A7BA83D10FE519"
```

```
HTTP/1.1 200 OK
Authorization-Info: nextnonce=
 "29FE72D109C7EF23841AB914F0C3B831"
 qop= "auth"
 rspauth="89F5A4CE6FA932F6C4DA120CEB754290"
 cnonce="CFA9207102EA210EA210FFC1120F6001110D073"
 ...
```

Qop (quality of protection enhancements) allows client-server negotiation on the security features to use (ex. Integrity protection also for message bodies)

DA: Advantages

- Password is not sent in cleartext anymore!
- Replay attack is not possible (at least if the nonce is randomly generated every time)
- A client can verify if the server is the same entity that generated the challenge (client nonce)

DA: Problems

- Different security profiles (for retro-compatibility with legacy version) may lead to insecure implementations
- Server's true identity cannot be verified! Indeed, we can only confirm that the server is the same entity that generated the challenge (man-in-the-middle attack still possible)
- MD5 algorithm is considered insecure nowadays