TURN REST Server API

draft-uberti-behave-turn-rest-00

Justin Uberti, Google

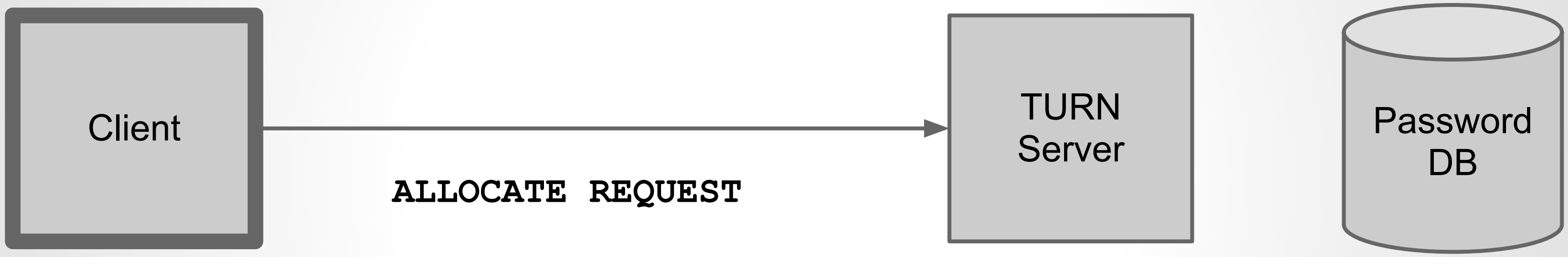
Typical TURN Auth: Config

Client
TURN Server
Password DB

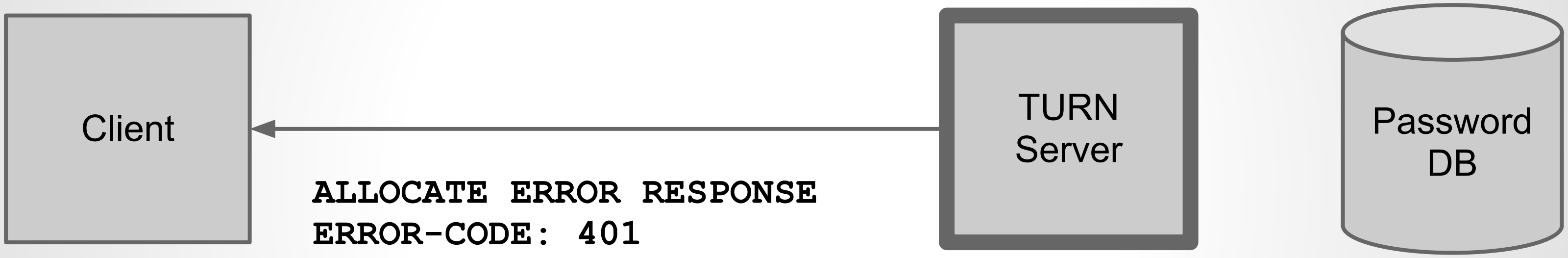
WebRTC JavaScript code:

```
var iceServer = {
  uris: ["turn:turn.bar.com:3478?proto=udp"],
  username: foo
  credential: mysecret
};
var config = {
  iceServers: [iceServer]
};
var pc = new PeerConnection(config, null);
```

Typical TURN Auth: TURN Request

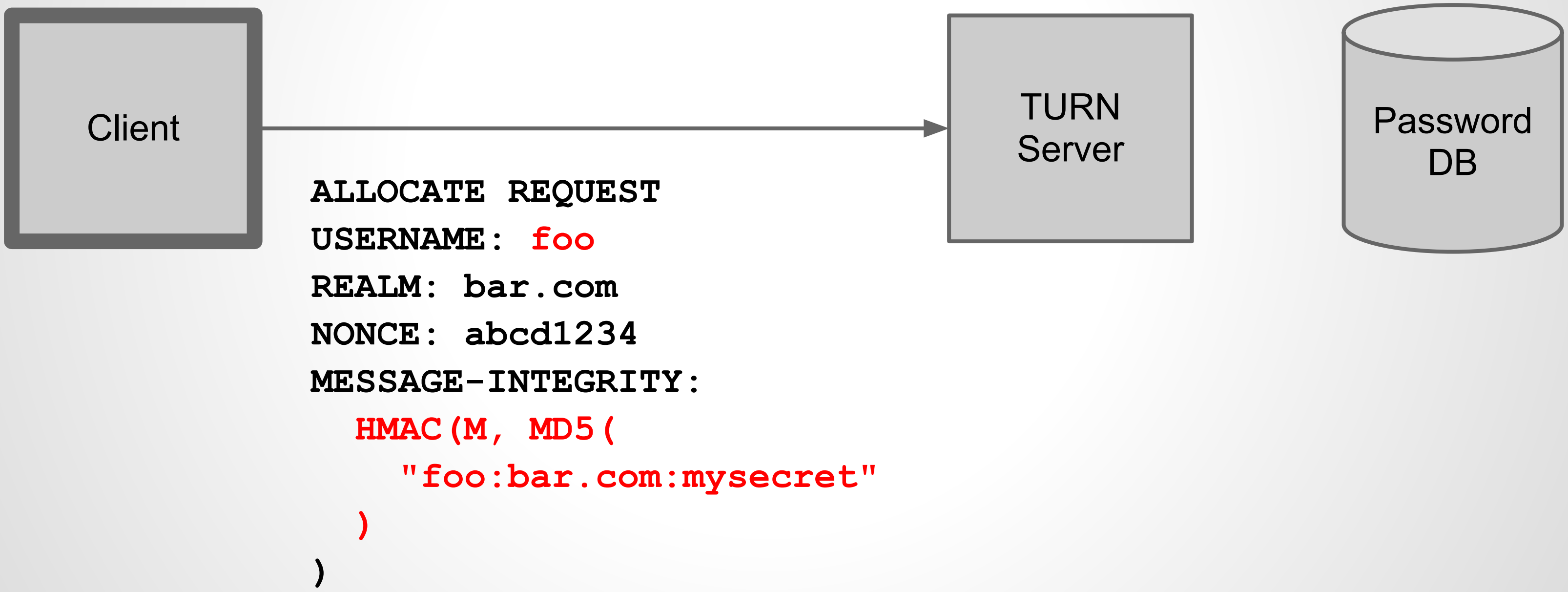


Typical TURN Auth: TURN Error Response



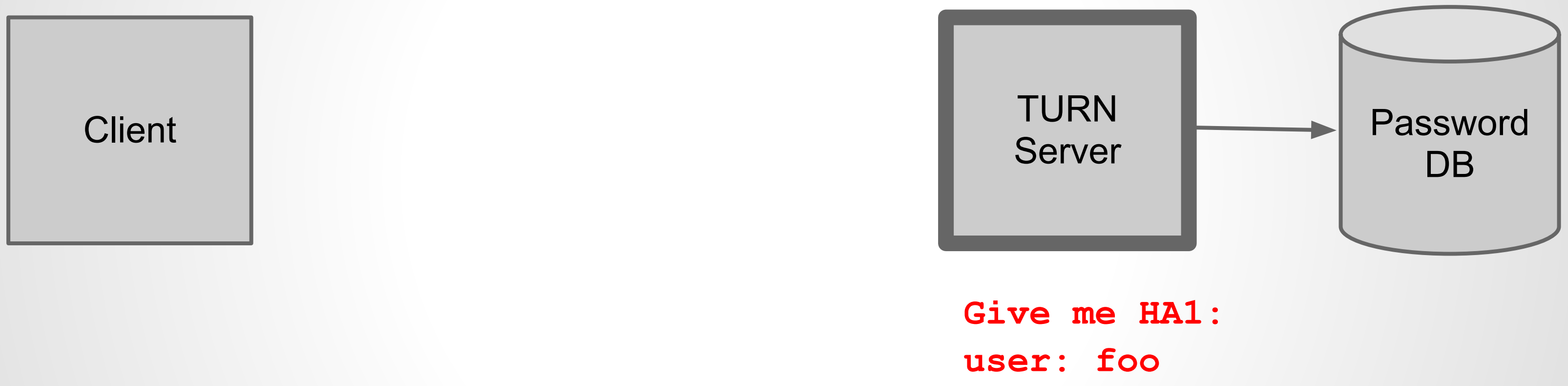
```
ALLOCATE ERROR RESPONSE
ERROR-CODE: 401
(Unauthorized)
REALM: bar.com
NONCE: abcd1234
```

Typical TURN Auth: TURN Request (2)

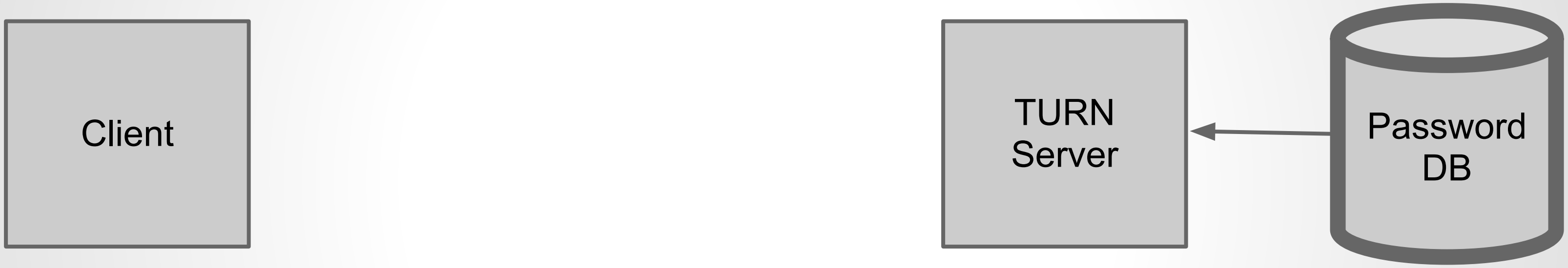


```
ALLOCATE REQUEST
USERNAME: foo
REALM: bar.com
NONCE: abcd1234
MESSAGE-INTEGRITY:
  HMAC(M, MD5(
    "foo:bar.com:mysecret"
  )
)
```

Typical TURN Auth: HA1 Request



Typical TURN Auth: HA1 Response



```
Here you go:
ha1: MD5("foo:bar.com:
mysecret")
```

Typical TURN Auth: Verify

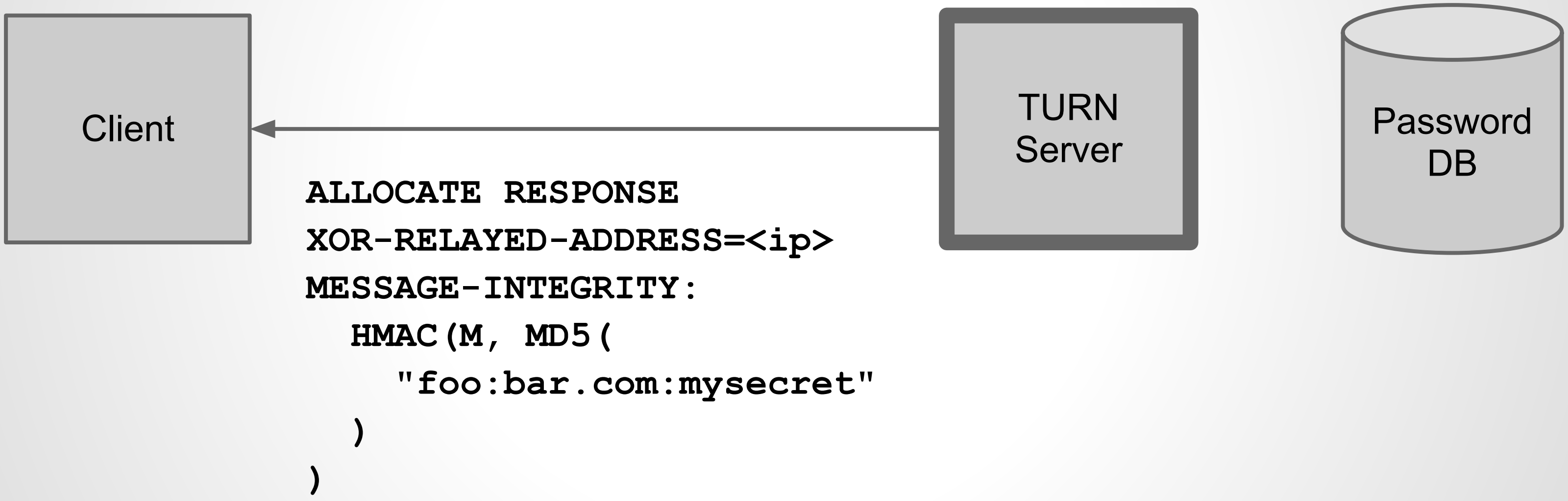
Client

TURN
Server

Password
DB

MESSAGE-INTEGRITY verify
against
HMAC(M, HA1)

Typical TURN Auth: TURN Response



```
ALLOCATE RESPONSE
XOR-RELAYED-ADDRESS=<ip>
MESSAGE-INTEGRITY:
  HMAC(M, MD5(
    "foo:bar.com:mysecret"
  )
)
```

Inherent Problems

The problems with the TURN long-term auth exchange are documented in ***draft-reddy-behave-turn-auth***

- TURN password must be kept secret (hard for WebRTC apps)
- TURN password vulnerable to offline dictionary attacks on MESSAGE-INTEGRITY
- TURN server must consult a password database to verify MESSAGE-INTEGRITY
- TURN username value is passed in the clear, can be used for traffic analysis

Proposed Solution

Client makes a HTTP request to a web service to get ephemeral (time-limited) credentials:

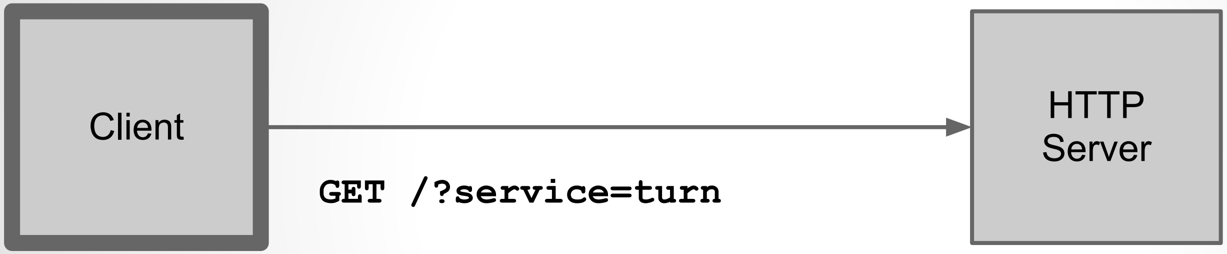
- No long-term credentials to keep secret; even if discovered, credential usefulness is limited
- Username contains no externally-identifying information
- Password is machine-generated, to prevent dictionary attacks
- Response also includes location of TURN server, avoiding complex SRV lookups

Credential Verification

While the TURN server could verify credentials against the HTTP server, the draft suggests a stateless design that **requires no backchannel.**

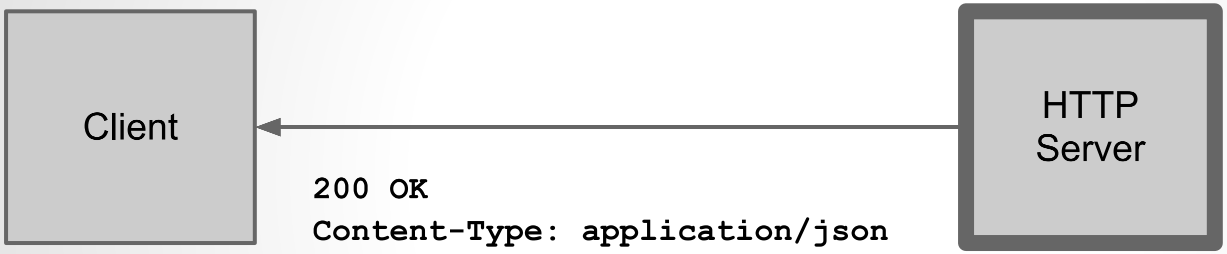
- Username is **credential expiration timestamp** + any desired application data
- Password is **HMAC(username, SS)**, where SS is a **shared secret key** between HTTP and TURN servers
- To get HA1, TURN server simply does MD5(<username>:<realm>:<hmac>)

Stateless TURN Auth: HTTP Cred Request



```
GET /?service=turn
```

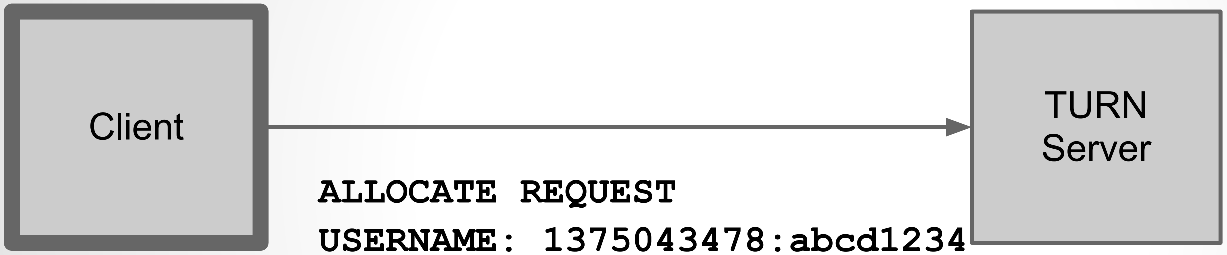
Stateless TURN Auth: HTTP Cred Response



```
200 OK
Content-Type: application/json

{
  username: "1375043478:abcd1234",
  password: <HMAC("1375043487:abcd1234", SS)>
  ttl: 86400,
  uris: [
    "turn:turn.bar.com:3478?proto=udp",
    "turn:turn.bar.com:3478?proto=tcp",
    "turns:turn.bar.com:443?proto=tcp"
  ]
}
```

Stateless TURN Auth: TURN Request (2)



```
ALLOCATE REQUEST
USERNAME: 1375043478:abcd1234
REALM: bar.com
NONCE: abcd1234
MESSAGE-INTEGRITY:
  HMAC(M, MD5(
    "1375043478:abcd1234:bar.com:<hmac-password>"
  )
)
```

Stateless TURN Auth: Verify

Client

TURN Server

1. Parse timestamp from USERNAME (1375043478)
2. Check that timestamp is in the future
3. Compute password:
   HMAC(1375043478:abcd1234, SS)
4. Compute HA1: MD5(1375043478:abcd1234:bar.com:
   <hmac-password>)
5. MESSAGE-INTEGRITY verify against
   HMAC(M, HA1)
6. If it's cool, return success response
7. No communication with HTTP server needed!

Why not Short Term Credentials?

- STUN defines a short-term credential mechanism, but this mechanism doesn't support nonces, opening the door for trivial replay attacks

Questions

- Adopt as WG draft?
- Propose generic HTTP mechanism + example stateless implementation, or focus exclusively on stateless design?