UMA Macaroons

Tagline

Macaroons, made from scratch using an UMA recipe with the fresh HMAC ingredients.

Introduction

Bearer tokens are vulnerable at rest and in transit when an attacker is able to intercept a token to illegally access private information. In order to mitigate some of the risk associated with bearer tokens, UMA Macaroons may be used instead of bearer tokens. UMA Macaroons are cryptographically chained blocks of data bearing a chronological tamper-resistant record of all their possessors and the changes that have been made to them. In the authorization flow, UMA Macaroons use a complex combination of Chained-MACs-with-Multiple-Messages and Chained-MACs-with-Multiple-Keys constructions as a correlation mechanism among all participants and their data. UMA Macaroons adopt the User-Managed Access concept of authorization server, resource server, client, resource owner and requesting party.

Key Differences from Google Macaroons

- Authenticated possessors.
- Claims are used instead of caveats.
- Different HMAC chaining.
- Verification at the authorization server.

Following we use the term *macaroon* to refer to UMA Macaroon.

Concept of MACs Chaining

The POCOP Token Mechanism is used to construct macaroons.

1. To ensure integrity protection of macaroon claims, the first macaroon uses a Chained-MACs-with-Multiple-Messages construction. All MACs must be discarded

after use.

$MAC_{macaroon\_1}$ = HMAC(...HMAC(HMAC($K_{possessor\_1}$, $claims\_1_{possessor\_1}$), $claims\_2_{possessor\_1}$), ...$claims\_n_{possessor\_1}$)

2. Chained-MACs-with-Multiple-Keys construction is used to assure the authenticity of macaroons. The input $MAC_{macaroon\_1}$ must be discarded after use. The final $MAC_{macaroon\_1}$ can be published, there is no need to hide it.

$MAC_{macaroon\_1}$ = HMAC($K_{possessor\_1}$, $MAC_{macaroon\_1}$)

- Hop to the possessor_2.

$MAC_{macaroon\_1}$ = HMAC($K_{possessor\_2}$, $MAC_{macaroon\_1}$)

3. The second macaroon uses the Chained-MACs-with-Multiple-Messages construction in a similar manner to the first macaroon. The $MAC_{macaroon\_1}$ is added to the possessor_2 macaroon in the first claims. The other MACs must be discarded after use.

$MAC_{macaroon\_2}$ = HMAC(...HMAC(HMAC($K_{possessor\_2}$, $MAC_{macaroon\_1}$), $claims\_2_{possessor\_2}$), ...$claims\_n_{possessor\_2}$)

To simplify notation, we use the Double HMAC construct – a nested HMAC function, denoted by DHMAC, that takes 3 inputs (*K*, *MAC*, *m*) and outputs a message authentication code *MAC* = DHMAC(*K*, *MAC*, *m*) = HMAC(*K*, HMAC(*MAC*, *m*)), where *K* is the secret key, *MAC* is the input message authentication code, and *m* is the message to be authenticated.

Macaroons possessors must be registered at the authorization server (public clients can use dynamic registration to become confidential clients). Macaroons are verified via the introspection endpoint of the authorization server.

Use Cases

Advanced authorization scenarios e.g. chained resource servers.

Example of Chained Macaroons

Each macaroon contains three mandatory claims:

- The random NONCE to prevent replay attack.
- The timestamp of when the macaroon was created.
- The URI that identifies who created the macaroon.

Additional groups of optional claims (e.g. in JSON format) can be added at any time until the macaroon is sent to the next possessor.

The HMAC chain may started with an AS or any other registered client.

- The AS is the first macaroon possessor.

  MAC_{AS} = HMAC(K_{AS}, $NONCE_{AS}$)

  MAC_{AS} = DHMAC(K_{AS}, MAC_{AS}, $Timestamp_{AS}$)

  MAC_{AS} = DHMAC(K_{AS}, MAC_{AS}, URI_{AS})

  MAC_{AS} = DHMAC(K_{AS}, MAC_{AS}, $claims\_1_{AS}$)

  ...

  MAC_{AS} = DHMAC(K_{AS}, MAC_{AS}, $claims\_n_{AS}$)
- Hop to the next possessor – the client.

  MAC_{client} = HMAC(K_{client}, $NONCE_{client}$)

  MAC_{client} = DHMAC(K_{client}, MAC_{client}, $Timestamp_{client}$)

  MAC_{client} = DHMAC(K_{client}, MAC_{client}, URI_{client})

  MAC_{client} = DHMAC(K_{client}, MAC_{client}, MAC_{AS})

MAC_{client} = DHMAC(K_{client}, MAC_{client}, $claims\_1_{client}$)

...

MAC_{client} = DHMAC(K_{client}, MAC_{client}, $claims\_n_{client}$)

- Hop to the next possessor – the RS_1.

MAC_{RS_1} = HMAC(K_{RS_1}, $NONCE_{RS\_1}$)

MAC_{RS_1} = DHMAC(K_{RS_1}, MAC_{RS_1}, $Timestamp_{RS\_1}$)

MAC_{RS_1} = DHMAC(K_{RS_1}, MAC_{RS_1}, URI_{RS_1})

MAC_{RS_1} = DHMAC(K_{RS_1}, MAC_{RS_1}, MAC_{client})

MAC_{RS_1} = DHMAC(K_{RS_1}, MAC_{RS_1}, $claims\_1_{RS\_1}$)

...

MAC_{RS_1} = DHMAC(K_{RS_1}, MAC_{RS_1}, $claims\_n_{RS\_1}$)

- Hop to the next possessor – the RS_2.

MAC_{RS_2} = HMAC(K_{RS_2}, $NONCE_{RS\_2}$)

MAC_{RS_2} = DHMAC(K_{RS_2}, MAC_{RS_2}, $Timestamp_{RS\_2}$)

MAC_{RS_2} = DHMAC(K_{RS_2}, MAC_{RS_2}, URI_{RS_2})

MAC_{RS_2} = DHMAC(K_{RS_2}, MAC_{RS_2}, MAC_{RS_1})

MAC_{RS_2} = DHMAC(K_{RS_2}, MAC_{RS_2}, $claims\_1_{RS\_2}$)

...

MAC_{RS_2} = DHMAC(K_{RS_2}, MAC_{RS_2}, $claims\_2_{RS\_2}$)

- The last MAC_{RS_2} can be verified via the introspection endpoint of the AS.

Nested Macaroon / Third-Party Claims

A macaroon can contain another macaroon.

Example of Nested Macaroon

This is an excerpt from the above Example of Chained Macaroons extended by third party claims.

...

- Hop to the next possessor – the client.

MAC_{client} = HMAC(K_{client}, $NONCE_{client}$)

MAC_{client} = DHMAC(K_{client}, MAC_{client}, $Timestamp_{client}$)

MAC_{client} = DHMAC(K_{client}, MAC_{client}, URI_{client})

MAC_{client} = DHMAC(K_{client}, MAC_{client}, MAC_{AS})

- Hop to the next possessor – the AS_third_party.

$MAC_{AS\_third\_party}$ = HMAC($K_{AS\_third\_party}$, $NONCE_{AS\_third\_party}$)

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, $Timestamp_{AS\_third\_party}$)

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, $URI_{AS\_third\_party}$)

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, MAC_{client})

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, $claims\_1_{AS\_third\_party}$)

...

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, $claims\_n_{AS\_third\_party}$)

- Hop to the next possessor – back to the client.

MAC_{client} = DHMAC(K_{client}, MAC_{client}, $MAC_{AS\_third\_party}$)

MAC_{client} = DHMAC(K_{client}, MAC_{client}, $claims\_1_{client}$)

...

MAC_{client} = DHMAC(K_{client}, MAC_{client}, $claims\_n_{client}$)

- Hop to the next possessor – the RS_1.

...

Confidential Claims

~~Third-party claims can be chained using the AES-GCM authenticated encryption algorithm instead of the HMAC message authentication algorithm.~~

This is an excerpt from the above Example of Nested Macaroon extended by confidential third party claims.

...

- Hop to the next possessor – the AS_third_party.

$MAC_{AS\_third\_party}$ = HMAC($K_{AS\_third\_party}$, $NONCE_{AS\_third\_party}$)

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, $Timestamp_{AS\_third\_party}$)

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, $URI_{AS\_third\_party}$)

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, MAC_{client})

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, Enc($K_{AS\_third\_party}$, $claims\_1_{AS\_third\_party}$))

...

$MAC_{AS\_third\_party}$ = DHMAC($K_{AS\_third\_party}$, $MAC_{AS\_third\_party}$, Enc($K_{AS\_third\_party}$, $claims\_n_{AS\_third\_party}$))

- Hop to the next possessor – back to the client.

...

Conclusion

(TBD)

Acknowledgment

Credits go to WG - User-Managed Access and Google Research Publications.