

HMAC security for Babel

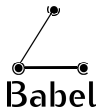
Juliusz Chroboczek

Joint work with Clara Dô and Weronika Kołodziejak

IRIF

Université Paris-Diderot (Paris 7)

17 July 2018



Disclaimer

I am just the janitor here. Credit goes to:

- Denis Ovsienko (started the whole thing);
- Clara Dô and Weronika Kołodziejak (implementation and new protocol design based on Denis' work);
- David Schinazi (pseudo-header);
- Markus Stenberg (index mechanism);
- Toke Høyland-Jørgensen;
- Florian Horn and Paul Rozière.

Two protocols for Babel security

There is a natural tension between:

- **simple**, reviewable protocol;
- useful **features**.

Two security protocols for Babel:

HMAC (this talk):

- Babel **unchanged**;
- **no dependencies**;
- minimal features:
 - symmetric keying;
 - few keys per interface;
 - **key rotation**.

DTLS (David's talk):

- **Babel over unicast**;
- **depends on DTLS**;
- features of DTLS:
 - **asymmetric keying**;
 - **pairwise keying**;
 - **ASN.1**.

Naive HMAC

HMAC guarantees the **authenticity** of a message.

Very roughly, send

$$m \parallel H(m \parallel k)$$

where m is the message, k is the (secret) key, and H is a cryptographic hash function (e.g. SHA256).

HMAC does **not protect against replay**.

B: announce, HMAC

A installs a route through B.

(later)

C: **replays** announce, HMAC.

A **installs a route through C**.

Protecting addresses: pseudo-header

Mitigation: **protect addresses** as well as packet contents.

Send:

$$m \parallel H(\text{src} \parallel \text{dst} \parallel m \parallel k)$$

where src is the source address and dst is the destination address. src || dst is the **pseudo-header**.

B: announce, HMAC

A: installs a route through B.

(later)

C: **replays** announce, HMAC, **with B's source address**.

A **installs a route** through B.

The protocol is **still vulnerable to replay**, but the consequences are less severe.

HMAC with simple replay protection

Include a **per-sender packet** counter in each packet.

B: announce, **PC=42**, HMAC

A installs a route through B.

(later)

C: **replays** announce, **PC=42**, HMAC

A: **rejects** C's packet (PC too old).

The protocol requires that:

- the sender's PC is **strictly increasing** (reliable hardware clock or reliable persistent storage);
- the receiver keeps **persistent state** about each neighbour **forever**.

Simple replay protection: issues

RFC 7298 does simple replay protection. Two issues:

- requires that every sender maintain a **strictly monotonic** PC;
- requires that every receiver keep the PC history of every peer it has ever encountered **forever**.

Persistent **sender state** difficult to do portably;

- **persistent storage** is non-portable and unreliable;
- **hardware clocks** are not universal, sometimes reset even when available.

Very unpleasant failure mode: node is blackholed.

Persistent **receiver state** (“ANM”) unrealistic:

- **unbounded** amounts of **persistent storage**.

Failure mode is gentler: after loss of state, we are vulnerable to replay (Denis Ovsienko, IETF 96).

Avoiding persistent state

In draft-do-babel-hmac, we **avoid persistent state** by using **two additional mechanisms**:

- a **challenge** mechanism (a cryptographic handshake, with **nonces**) to create fresh receiver state;
- an **index** mechanism to indicate when sender state has been reset.

Receiver-side state: challenge with nonce

B: announces, PC=42, HMAC

A (has no state about B): challenge, nonce=57, HMAC

B: challenge reply, nonce=57, PC=43, HMAC

Since the nonce is fresh, B's challenge reply cannot be a replayed packet. A can safely establish receiver state from the challenge reply.

A can send a new challenge at any time (cost = 1RTT); when to purge receiver state is now an implementation detail.

Sender-side state: indices

B: announce, PC=42, **index=99**, HMAC

A (index mismatch): challenge, nonce=57, HMAC

B: challenge reply, nonce=57, PC=43, **index=99**, HMAC

Whenever it has **no state** (e.g. at boot), B generates a **fresh index**, which it sends with every PC.

Whenever it detects an **index change**, A sends a new **challenge**; if the challenge is successful, A discards its old state about B.

B can generate a **new index at any time** (cost = 1RTT); when to purge sender state is now an **implementation detail**.

Specification and implementation status

draft-do-babel-hmac-00 needs some more work, but **good enough** for implementation. **16 pages** including boilerplate, 7 normative.

We already have **two independent implementations**:

- Dô, Kołodziejak:
 - integrated in babeld (but not merged yet);
 - 793 lines of code (not counting SHA256);
 - minimal changes to babeld;
 - “Easier to implement than RFC 7298.”
- Højland-Jørgensen:
 - integrated in BIRD (but not merged yet);
 - 533 lines of code;
 - “I found the draft pleasant to read and straightforward to implement.”

Open questions

A number of **choices** remain (already discussed on list):

- **size** of nonce and index;
- use of **packet trailer**;
- explicit vs. **implicit indices**.

Open questions: size of nonce and index

A Babel TLV can be up to 255 octets long. This gives enough space for up to **255 octets of nonce** and up to **251 octets of index**.

- a nonce can usefully encode a cookie (for DoS avoidance); arbitrary size nonces do not complicate implementation much;
- we see no reason to allow large indices; arbitrary size indices complicate implementation somewhat.

Proposition:

- **nonces can be any size**, from 0 to 255 octets;
- **indices can be 0 to 10 octets**, larger indices are silently dropped.

Open questions: packet trailer

Where should HMACs live? Two possibilities:

- within the **packet trailer**, beyond the packet body;
- within the **packet body** itself.

Storing HMACs within the **packet body** itself (which participates in HMAC computation) requires a **complex dance** of clearing the HMAC TLV before computing HMAC.

Storing HMACs in the packet trailer **simplifies implementation**, at the cost of a **slightly more complex specification**. The authors support this choice.

Open questions: implicit indices

Markus Stenberg noticed that **the index is already known** by the receiver, except during challenges.

Idea: **omit the index** in ordinary packets, only **include it in challenge replies**.

Pros:

- **shaves off a few bytes** on the wire;
- **clever** solution.

Cons:

- HMAC can only be **computed after parsing** the packet;
- more **complex encoding**;
- **clever** solution.

See Appendix C of draft-do-babel-hmac-00 for details.

Conclusion

A **simple**, easily **implementable** cryptographic **authenticity extension** for Babel:

- **simple specification**: 16 pages (7 normative);
- **simple implementation**: 533 to 793 lines of code;
- two **independent**, interoperable **implementations**.

Some open issues, need discussion and thought.

Please adopt?