Universidade do Minho

Mestrado em Engenharia Informática

Tecnologia Criptográfica

Auditing a Poly1305 MAC implementation in Jasmin for x86

Miguel Miranda Quaresma
A77049

December 28, 2018

Abstract

Poly1305 is a one time authenticator that generates a message authentication code for a given input and secret key using, for that purpose, a similar mechanism to universal hashing. Jasmin is a framework for developing high performance and high assurance cryptographic software. The present works aims to audit an implementation of the Poly1305 MAC using the Jasmin framework. I'll begin by describing the Poly1305 MAC at a high(abstraction) level, followed by an in-depth analysis of the Jasmin implementation of the algorithm. The work concludes with an auditing/verifcation of the premisses/assumptions that were made in the implementation.

1 Introduction

Message authentication codes play a major role in guaranteeing the authenticity and integrity of data being sent across an untrusted channel. There are many cryptoprimitives that implement MAC's and, for a long time, HMACs(Hash-MACs) were preferred over others due to their performance, with other primitives such as the ones based on universal hashing being discarded. This preference was further reinforced by the introduction of assembly instructions to perform hash functions directly in hardware [2], such as Intel's instruction for SHA-256: `sha256rnds2`. The development of cryptoprimitives such as Poly1305 MAC using Jasmin, a framework for developing high performance and high assurance cryptographic software [1], allowed this tendency to be reversed by obtaining highly performant implementations with relative ease.

2 Poly1305 explained

Poly1305 is a message authentication code(MAC) that guarantees integrity and authenticity of messages. It achieves so similarly to a universal hash function, using a polynomial over a prime field to calculate a MAC for a given message,key pair. As stated Poly1305 evaluates a polynomial over a prime field, the prime, $2^{130} - 5$, being where the name (Poly**1305**) stems from. Thus, Poly1305 can be expressed by the following expression:

$$mac = (m1 * r^4 + m2 * r^3 + m3 * r^2 + m4 * r + k) \mod p$$

where mi is the i-th block of the message. Being an authenticator, Poly1305 also takes in a 256 bit secret, derived via a Password-Based Key Derivation Function(PBDKF), that is then split up into two blocks of 128 bits each, the first block being used for the parameter k, and the second for the parameter r. Poly1305 works by breaking the input in 16 byte blocks, appending each block with a 1 byte(00000001) to prevent forgery. It then proceeds to apply the following algorithm/formula:

```
h = 0
for block in blocks:
    h += block
    h *= r
mac = (h + k) mod 2^130-5
```

where:

- `h` is a (temporary) accumulator for the successive multiplications/additions
- `block` is a 16 byte message block plus a 1 byte
- `r` and `k` are the 128 bit values derived from the 256 bit secret

One of the consequences of the presented algorithm is the fact that, after a certain number of blocks, the variable h will overflow due to the successive multiplications. Therefore, it is necessary to perform the calculation via modular arithmetic. To do this in an efficient manner the prime $2^{130}-5$ was chosen to perform schoolbook multiplication futhermore, carry propagation is delayed to allow for fast(er) modular reduction.

2.1 (Schoolbook) Multiplication

After adding the message block mi to the current h value, h is multiplied with r using schoolbook multiplication. Note that, to perform this multiplciation, r and h are divided in five 4 byte(32 bit) blocks, called limbs , and multiplied with eachother as follows:

	h5	h4	h3	h2	h1
*	r5	r4	r3	r2	r1
	h5*r1	h4*r1	h3*r1	h2*r1	h1*r1
+	h4*r2	h3*r2	h2*r2	h1*r2	**5*h5*r2**
+	h3*r3	h2*r3	h1*r3	**5*h5*r3**	**5*h4*r3**
+	h2*r4	h1*r4	**5*h5*r4**	**5*h4*r4**	**5*h3*r4**
+	h1*r5	**5*h5*r5**	**5*h4*r5**	**5*h3*r5**	**5*h2*r5**

The expressions in bold are the modular reductions of the values overflowing 130 bits. This is possible due to the fact that 2^{130} is congruent with 5: **i.e**

$$2^{130} \equiv 5 \pmod{2^{130}-5}$$

hence why choosing a prime that is of the form 2^n-q, with q being a small number(such as $2^{130}-\mathbf{5}$), is important because it allows modular reduction to be performed by multiplying by 5 (**n.b.** this isn't a full modular reduction by $2^{130}-5$, this one can be delayed until the very end of the MAC calculation).

2.2 Limb size

As explained before (2.1), h and r are divided in 32 bit blocks called limbs and subsequently multiplied via schoolbook multiplication. There is a caviat in this process, if a h limb spans 32 bits, a multiplication will cause an overflow of a 64 bit register thus, it will make it imperative to perform carry propagation in each multiplication, hindering performance significantly. To prevent this from happening, Poly1305 clears 22 bits from r in the following manner:

```
r[0] = r[0] & 0x0fffffff
r[2] = r[2] & 0x0ffffffc
r[3] = r[3] & 0x0ffffffc
r[4] = r[4] & 0x0ffffffc
```

Thus each r limb will have, at most, 28 bits set (32-4). As a consequence, even if h limbs span 32 bits, a multiplication will only take up 60 bits (28+32).

After processing the entire message(**i.e** all it's blocks) the value k is added and a final full modular reduction is performed:

$$mac = (h+k) \pmod{2^{130}-5}$$

and the result is the MAC of the message.
The full modular reduction is just a matter of checking whether $h+k$ exceeds $2^{130}-5$, and subtracting $2^{130}-5$ if it is.

3 Jasmin Poly1305 Implementation

Let's now examine an implementation of the Poly1305 using Jasmin. Jasmin is a framework for developing cryptographic software inspired by qhasm, however Jasmin uses Coq to formally verify the (assembly) code generated by the Jasmin compiler, providing high *assurance* high performance cryptograhic code [1].

4 Conclusion

References

[1] José Bacelar Almeida et al. "Jasmin: High-Assurance and High-Speed Cryptography". In: Oct. 2017, pp. 1807–1823. DOI: 10.1145/3133956.3134078.

[2] Sean Gulley et al. *Intel ® SHA Extensions New Instructions Supporting the Secure Hash Algorithm on Intel ® Architecture Processors*. URL: https://software.intel.com/sites/default/files/artic 2013.