On Weak Keys and Forgery Attacks Against Polynomial-based MAC Schemes

Gordon Procter and Carlos Cid

Information Security Group,
Royal Holloway, University of London

Our Contributions

1. Study the underlying algebraic structure of polynomial-evaluation MACs and hash functions
2. Present a generalised forgery attack that:
   - extends Cycling Attacks (from FSE 2012)
   - describes all existing attacks against GCM
   - leads to a length extension attack against GCM
3. Identify many weak key classes for polynomial-based MAC constructions
   - almost every subset of the keyspace is weak

Overview

Overview

Polynomial-Evaluation-Based Hash Functions

Consider a message containing ciphertext, additional authenticated data and message length:

$$M = (M_1, \ldots, M_m) \in \mathbb{K}^m$$

The hash function family $\mathcal{H} = \{h_H : \mathbb{K}^\star \to \mathbb{K} | H \in \mathbb{K}\}$ is defined by a polynomial:

$$h_H(M) = \sum_{i=1}^{m} M_i H^i \in \mathbb{K}$$

This family is used for performance and low collision probabilities

Message Authentication

We can use $\mathcal{H}$ to construct fast and secure MACs
The authentication tag is the encryption of the hash, perhaps:

$$\mathrm{MAC}_{H||k}(M) = E_k(N) + h_H(M)$$

or

$$\mathrm{MAC}_{H||k}(M) = E_k(h_H(M))$$

In both cases:

Hash collision $\Rightarrow$ MAC forgery

Real Examples

GCM [MV05]

- Field: $\mathbb{K} = \mathbb{F}_{2^{128}}$
- Hash key: $H = E_k(0)$
- Tag encryption: Additive

CWC [KVW03]

- Field: $\mathbb{K} = \mathbb{F}_{2^{127}-1}$
- Hash key: $H = E_k(110^{126})$
- Tag encryption: Both

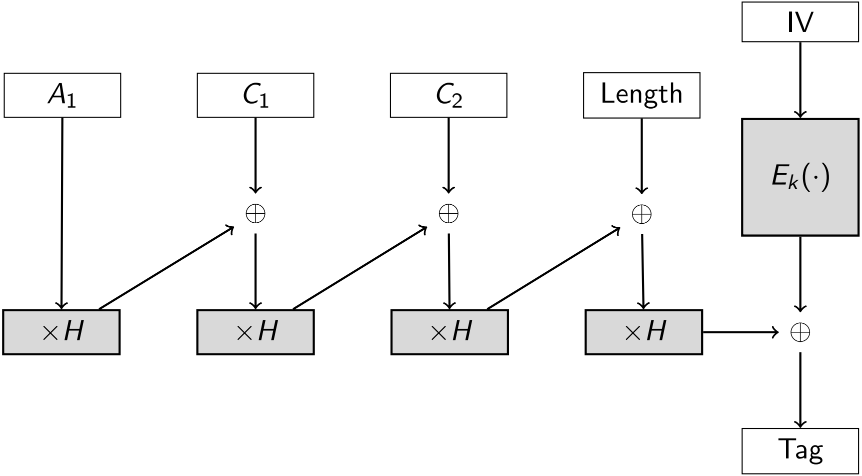
Poly-1305 [B05]

- Field: $\mathbb{K} = \mathbb{F}_{2^{130}-5}$
- Hash key: 128 bits (some specific bits zero)
- Tag encryption: Additive

SGCM [S12]

- Field: $\mathbb{K} = \mathbb{F}_{2^{128}+12451}$
- Hash key: $H = E_k(0)$
- Tag encryption: Additive

GCM's MAC



Overview

Adversary Model

The adversary can:

- Obtain T for (N, M) of his choosing
  - but can't repeat nonces
- Ask whether (N, M, T) is valid

Goal:

- Find (N, M, T) that is valid - without querying (N, M)

One Method:

1. Obtain T for (N, M)
2. Find M' with $h_H(M) = h_H(M')$
3. Then (N, M', T) is valid

Algebraic Background

Let H be the (unknown) hash key.
Suppose $q(x) = q_1 x + q_2 x^2 + \cdots + q_r x^r$ and that $q(H) = 0$

$$
\begin{aligned}
\text{Then } h_H(M) &= \sum_{i=1}^{m} M_i H^i \\
&= \sum_{i=1}^{m} M_i H^i + \sum_{i=1}^{r} q_i H^i \\
&= \sum_{i=1}^{r} (M_i + q_i) H^i \quad \text{(zero pad the shorter of } M \text{ and } q\text{)} \\
&= h_H(M + Q) \quad (Q = q_1 || \ldots || q_r, \text{ blockwise addition})
\end{aligned}
$$

Generalised Forgery

- We can find a hash collision by finding $q(x) = q_1 x + q_2 x^2 + \ldots + q_r x^r$ such that $q(H) = 0$
  - Hash collision $\Rightarrow$ MAC forgery

MAC forgery

Suppose we know that (N, M, T) is valid, then:

$$\begin{aligned} (N, M + Q, T) \text{ valid } &\Leftrightarrow q(H) = 0 \\ &\Leftrightarrow H \in \{x \in \mathbb{K} | q(x) = 0\} \end{aligned}$$

Similar observation made in [HP08]

Choosing $q(x)$

- Choosing $q(x)$ is difficult
  - we don't know H, so we don't know whether $q(H) = 0$
- Forgery Probability: $\frac{\#\text{roots of } q}{|\mathbb{K}|}$
- Want $q(x)$ with many roots:
  - high degree
  - no repeated roots

'The Naïve Approach'

Consider $\mathcal{D} \subseteq \mathbb{K}$, then:

$$q(x) = \prod_{\substack{H_i \in \mathcal{D} \\ \text{or } H_i = 0}} (x - H_i)$$

Examples of $q(x)$

All known attacks against GCM can be described in terms of the $q(x)$ that are used in the attacks

Ferguson: Attacks GCM when used with short tags

- Uses linearised polynomials
- Relies on linearity of squaring in $\mathbb{F}_{2^{128}}$
  - $q(x)$ 'looks like' $x + x^2 + x^4 + \ldots + x^{2^{17}}$
  - can keep track of roots using a matrix

Joux: Attacks GCM when nonces are repeated

- Need (N, M, T) and (N, M', T') valid (same N)
  - then $h_H(M) + h_H(M') = T + T'$
  - so $\underbrace{h_H(M + M') - (T + T')}_{\frac{q(H)}{H}} = 0$

Examples of $q(x)$

Saarinen: looks for subgroups of $\mathbb{F}_{2^{128}}$, so H with $H^t = 1$

- $H^t = 1 \Rightarrow H^{t+1} = H \Leftrightarrow \underbrace{H^{t+1} - H}_{q(H)} = 0$

- $$\begin{aligned} h_H(M) &= M_1 H + \ldots + M_{t+1} H^{t+1} + \ldots + M_m H^m \\ &= M_{t+1} H + \ldots + M_1 H^{t+1} + \ldots + M_m H^m \\ &= h_H(M') \end{aligned}$$

- Suggested fix:
  - use $\mathbb{F}_{2^{128}+12451}$: very few H with $H^{t+1} = H$

Targeted-Bit Forgeries

It may be useful to have some control over the message that is forged So far we know that $M_i \to M_i + q_i$, for example:

- If M_i is additional authenticated data, then we know the value of the authenticated data in the forged message
- If $\mathrm{Char}(\mathbb{K}) = 2$ and $M_i = P_i + E_k(\mathrm{CTR})$ is counter mode encrypted ciphertext, then we know that $P_i \to P_i + q_i$

We can do better:

$$q(H) = 0 \Leftrightarrow \alpha q(H) = 0 \quad \forall \alpha \in \mathbb{K} \setminus \{0\}$$

- $M_i \to M_i + \alpha q_i$: we can choose any α we like
- For one message block, we can choose the value of $M_i + \alpha q_i$

Similar observation made in [S12]

Length Extension Against GCM

In GCM:

$$M = \texttt{length}||A_1||\dots||A_a||C_1||\dots||C_p$$

`length` is only used to compute the hash (it's not sent)

1. Pick a forgery polynomial $q(x)$
2. Find the value of $M_1 = \texttt{length}_M$ in the valid message
   - it correctly encodes the length of the message
3. Find the length of $(M + \alpha Q)$
   - we know M and Q
4. Choose $\alpha \in \mathbb{K}$:
   - so that $\texttt{length}_M \to \texttt{length}_M + \alpha q_1 = \texttt{length}_{M+\alpha Q}$

Length Extension Against GCM

- With a cycling attack:
  - best we can do is a success probability of $\frac{m}{|\mathbb{K}|}$
  - m is the length of the message in the valid (Message, Tag) pair
- Now we can increase the length of the message:
  - can achieve better success probabilities
  - with much shorter valid (Message, Tag) pair
- Now we have a success probability $\frac{\max\{m\}}{|\mathbb{K}|}$
  - $\max\{m\}$ is the *maximum permissible* message length
    - as in original security proofs for GCM

Overview

Weak Keys

The identification of *weak keys* is an important part of the security assessment of any scheme.

Definition [HP08]

A set of keys $\mathcal{D}$ for a MAC algorithm is weak if:

- Forgery probability higher than otherwise expected
- Use can be detected:
  - by trying $< |\mathcal{D}|$ keys, and
  - using $< |\mathcal{D}|$ tag verification queries

Known Weak Keys

Handschuh and Preneel 2008

- $\mathcal{D} = \{0\}$ is weak
- Because $h_0(M) = 0 \quad \forall M$

Saarinen 2012

- $\mathcal{D}_t = \{H | H^t = 1\}$ is weak
- Can swap M_i and $M_{i+\lambda t}$ to detect

New Weak Key Classes

We show that almost every subset of the keyspace is weak (for any hash function based on polynomial evaluation), in particular:

$\mathcal{D}$ is weak if:

- $|\mathcal{D}| \geq 3$
- $|\mathcal{D}| \geq 2$ and $0 \in \mathcal{D}$

Method

Requires 1 valid tag, ≤ 2 verification queries

1. Test if $H \in \mathcal{D} \cup \{0\}$
2. Test if $H = 0$, if necessary

Consequences

- These are properties of all polynomial hashes
  - *not* specific to GCM
- No ‘safe’ fields
  - SGCM not much better
  - does protect against some methods of finding good $q(x)$
- It is well known that message length is important
  - *maximum permissible* message length is what matters
  - also the size of the field is important
- All polynomial evaluation hashes have many weak keys
  - maybe it’s better to talk of an unavoidable property from the algebraic structure, rather than the number of weak keys?
  - does having lots of weak keys make the algorithm weak?

The End - Thank You

- These are properties of all polynomial hashes
  - *not* specific to GCM
- No ‘safe’ fields
  - SGCM not much better
  - does protect against some methods of finding good $q(x)$
- It is well known that message length is important
  - *maximum permissible* message length is what matters
  - also the size of the field is important
- All polynomial evaluation hashes have many weak keys
  - maybe it’s better to talk of an unavoidable property from the algebraic structure, rather than the number of weak keys?
  - does having lots of weak keys make the algorithm weak?