

# **A leakage-resilient MAC**

Joachim Schipper

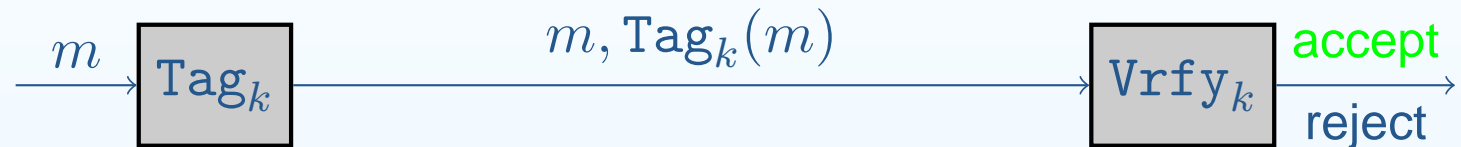
supervised by Eike Kiltz and Krzysztof Pietrzak

February 16, 2010

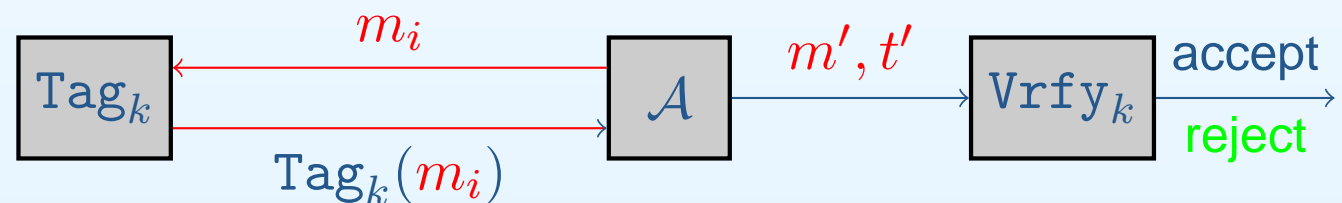
## Standard MACs

- Standard MACs
- Motivation
- Stateful MACs
- Results
- The construction

Message authentication codes (MACs) are used to detect changes in messages. Let  $k$  be a key.



$\text{Vrfy}_k$  typically calculates  $\text{Tag}_k(m)$  and compares it to the received tag.

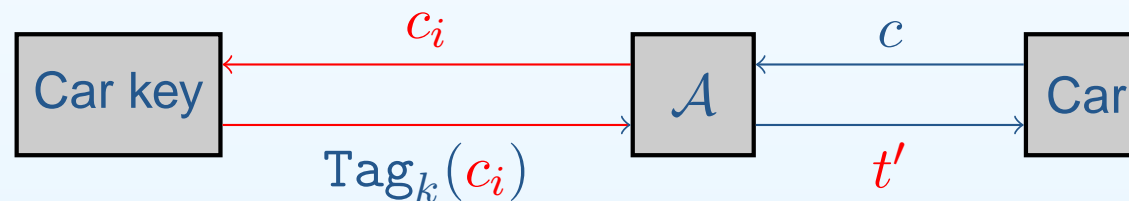


The adversary “wins” if  $\text{Vrfy}_k$  outputs “accept” for a message  $m' \notin \{m_1, m_2, \dots, m_n\}$ .

## Motivation

- Standard MACs
- **Motivation**
- Stateful MACs
- Results
- The construction

MACs can be used to authenticate devices, too: the verifier sends a random cookie and checks the returned tag. This is secure if the MAC is secure, assuming the adversary does not “have the key in hand”.

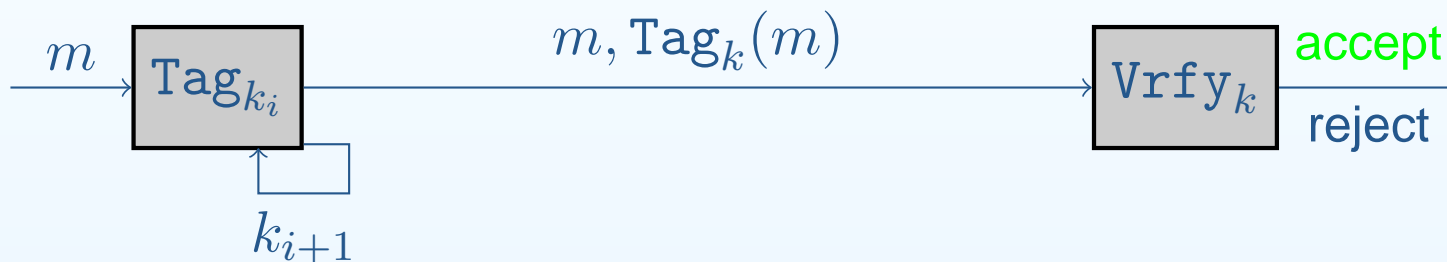


Of course, as the KeeLoq analysis ([EKM<sup>+</sup>08]) showed, leakage may allow an adversary to obtain the secret key  $k$ . We can't use the same key each round.

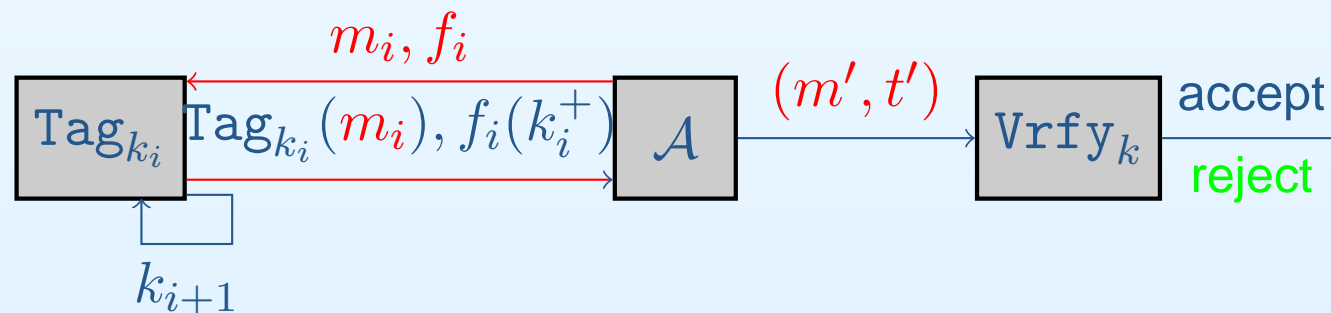
# Stateful MACs

- Standard MACs
- Motivation
- **Stateful MACs**
- Results
- The construction

Using the same key for each message doesn't work with general leakage (the adversary may learn a couple of bits each round). Stateful MACs are just regular MACs that are allowed to update their key each round.



We can now hope for security even with leakage.



## Results

- Standard MACs
- Motivation
- Stateful MACs
- **Results**
- The construction

We have constructed a leakage-resilient (stateful) MAC from a regular one-time MAC and a leakage-resilient stream cipher (e.g. [DP08] or [Pie09]) in “tree mode”. This MAC is quite useful in the situation just sketched (leaky key wants to authenticate itself).

- Efficient: 3 AES invocations to create a tag,  $O(\log(q))$  to verify it.
- Secure:  $P[\textit{broken}] \approx P[\textit{cipher broken}] + 2^\lambda P[\textit{MAC broken}]$ .

- Standard MACs
- Motivation
- Stateful MACs
- Results
- **The construction**

## The construction

We simply define

$\text{Tag}_{k_i}(m) = \{(X_{i+1}, k_{i+1}) \leftarrow S(k_i); \text{return Tag}_{X_{i+1}}(m)\}$ . Why does this work?

- If  $\text{Tag}_k(\cdot)$  is an  $\varepsilon$ -secure MAC, then  $\text{Tag}_{X_i}(\cdot)$  is  $2^\lambda \varepsilon$ -secure if  $X_i$  has pseudoentropy  $\log |X_i| - \lambda$ .
- Due to the use of a leakage-resilient stream cipher, the pseudoentropy of  $X_i$  is actually (nearly)  $\log |X_i| - \lambda$ , even with leakage.
- A “tree” version of [DP08] or [Pie09] can be used to get a stateless verifier with about  $\log(q)$  AES invocations.

Full details will appear in my thesis, “A leakage-resilient MAC”. Ask me for a copy sometime this week or search the web in a week or two.

- Standard MACs
- Motivation
- Stateful MACs
- Results
- The construction

## References

- [DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.
- [EKM<sup>+</sup>08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the power of power analysis in the real world: A complete break of the KeeLoq code hopping scheme. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2008.
- [Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 462–482. Springer, 2009.