McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes

Ewan Fleischmann   Christian Forler   Stefan Lucks

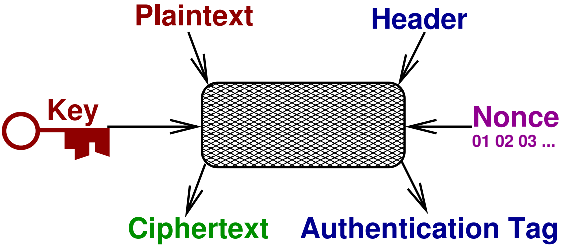
Bauhaus-Universität Weimar

FSE 2012

Overview

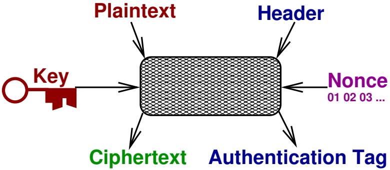
1. Motivation

- Goldwasser and Micali (1984):
  requirement: given 2 ciphertexts, adversary cannot even detect when the same plaintext has been encrypted twice
  consequence: encryption stateful or probabilitistic (or both)
- Rogaway (FSE 2004): formalizes state/randomness by **nonces**



Authenticated Encryption

- first studied by Katz and Young (FSE 2000) and Bellare and Namprempre (Asiacrypt 2000)
- since then many proposed schemes,
- nonce based,



- and proven secure assuming a **“nonce-respecting adversary”**
- any implementation allowing a **nonce reuse** is not our problem . . . but maybe it should

Nonce Reuse in Practice

- IEEE 802.11 [Borisov, Goldberg, Wagner 2001]
- PS3 [Hotz 2010]
- WinZip Encryption [Kohno 2004]
- RC4 in MS Word and Excel [Wu 2005]
- ...

application programmer mistakes:

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

other issues:

- restoring a file from a backup
- cloning the virtual machine the application runs on
- ...

Nonce Reuse – what to Expect?

our reasonable (?) expectations

- some plaintext information leaks:
  - identical plaintexts
  - common prefixes
  - ect.
- but not too much damage:
  1. authentication not affected
  2. no immediate plaintext recovery

Nonce Reuse – what Really Happens!

a double-disaster for almost all current AE schemes

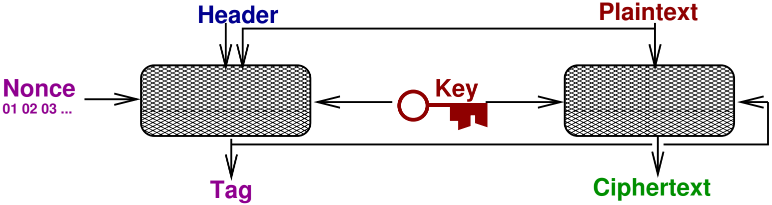
1. forgeries
2. plaintext recoveries (often like "one-time-pad used twice")

Systems with Protection from Nonce Reuse

SIV (Rogaway, Shrimpton, Eurocrypt 06) and similar schemes

Sequentially execute the following two steps:

1. generate authentication tag (from nonce, header, plaintext)
2. encrypt plaintext, using tag as "syntetic" nonce



Properties of SIV and its Fellows?

security under nonce reuse meets our "reasonable expectations":

authenticity: not affected!

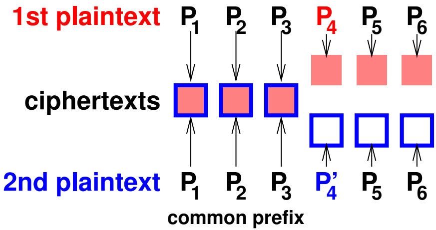
privacy: leaks whether two plaintexts are equal, but not more

but **inherently off-line** (user must read entire plaintext twice):

- high latency (first bit of ciphertext can only be sent after last bit of plaintext has been read)
- storage issues (enjoy encrypting your harddisk backup ...)

2. The McOE Approach

on-line permutations
(Bellare et al., Crypto 01):



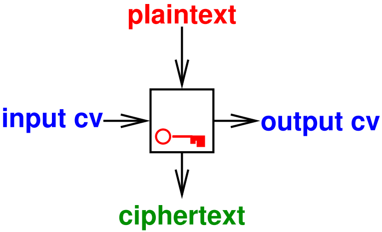
security under nonce reuse still meets "reasonable expectations":

authenticity: not affected!

privacy: leaks ~~whether two plaintexts are equal~~ the length of common plaintext prefixes, but not more

Our Main Tool: Chaining Blockciphers

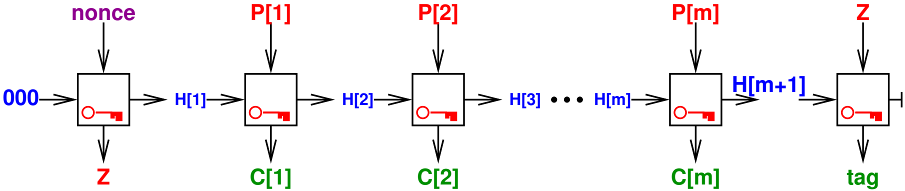
for the moment, assume we actually have such primitives



- like a block cipher
- two additional parameters:
  - "input chaining value"
  - "output chaining value"

- for fixed **input cv** **good block cipher** (PRP)
- regarding the **output cv** **good keyed hash function**:
  - weak collision resistance
    (hard to find two input pairs with colliding **output cv**s)
  - weak preimage resistance
    (hard to find an input pair with **output cv** $=$ **000**)

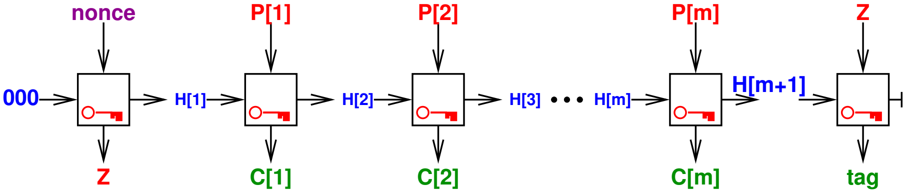
McOE



1. Encrypt **nonce**, using **000**, to generate **H[1]** and secret **Z**.
2. For **i** in 1, . . . , *m*:

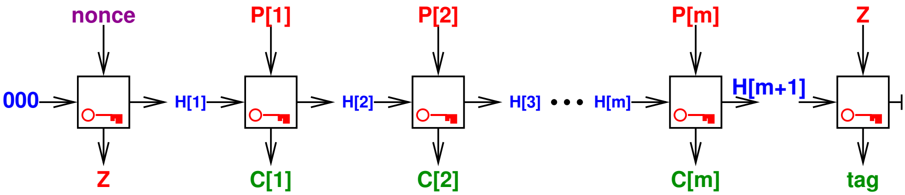
   encrypt **P[i]**, using **H[i]**, to generate **H[i+1]** and **C[i]**.
3. Encrypt **Z**, using **H[m+1]**, to generate the authentication **tag**.

3. Why is this secure? (Some Intuition)



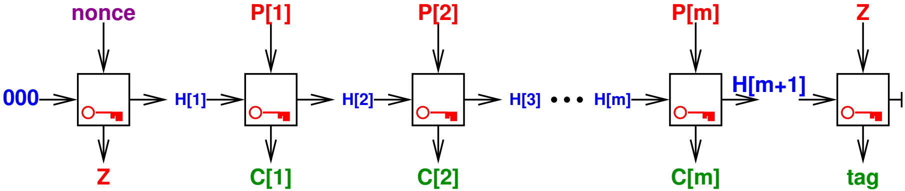
1. **nonce**-reuse: an Ind-CPA secure OPerm ($\rightarrow$ next slide) (common plaintext prefixes $\leftrightarrow$ common ciphertext prefixes)

3. Why is this secure? (Some Intuition)



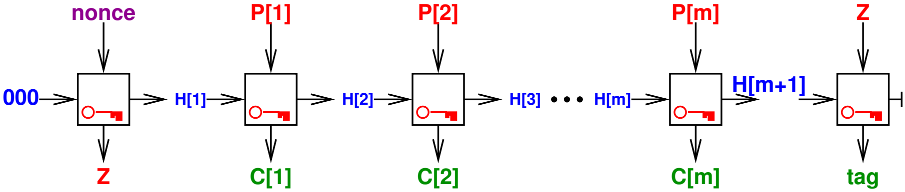
1. **nonce**-reuse: an Ind-CPA secure OPerm ($\rightarrow$ next slide)
   (common plaintext prefixes $\leftrightarrow$ common ciphertext prefixes)
2. **nonce**-respecting: Ind-CPA secure
   (different **nonce**s make common plaintext prefixes disappear)

3. Why is this secure? (Some Intuition)



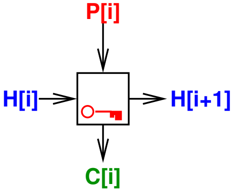
1. **nonce**-reuse: an Ind-CPA secure OPerm ($\rightarrow$ next slide) (common plaintext prefixes $\leftrightarrow$ common ciphertext prefixes)
2. **nonce**-respecting: Ind-CPA secure (different **nonce**s make common plaintext prefixes disappear)
3. Int-CTXT secure: A forger would need to predict **tag**, the encryption of **Z** using **H[m+1]**. But **Z** is secret.

Nonce-Reuse: Ind-CPA-secure OPerm (1)



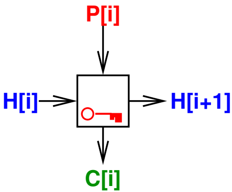
- Consider a query (**nonce**, **P[1]**, ..., **P[m]**).
- Let **i** $\in \{1, \ldots, m\}$ be the smallest index, such that there is no other query (**nonce**, **P[1]**, ..., **P[i]**, ...) with the same nonce and **i** blocks of prefix.
- **H[i]** is uniquely determined.

Nonce-Reuse: Ind-CPA-secure OPerm (2)



H[i] is given, **P[i]** is new. Exploit the properties of the chaining bc:

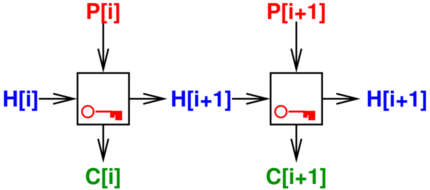
Nonce-Reuse: Ind-CPA-secure OPerm (2)



H[i] is given, **P[i]** is new. Exploit the properties of the chaining bc:

- **Good block cipher:** **C[i]** is like a random value.
- **Good keyed hash function:** **H[i+1]** has never been used before as an **input cv**.

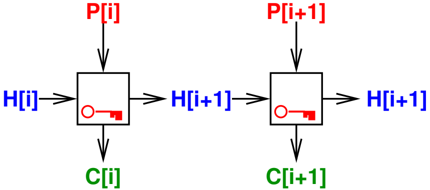
Nonce-Reuse: Ind-CPA-secure OPerm (2)



H[i] is given, **P[i]** is new. Exploit the properties of the chaining bc:

- **Good block cipher:** **C[i]** is like a random value.
- **Good keyed hash function:** **H[i+1]** has never been used before as an **input cv**.

Nonce-Reuse: Ind-CPA-secure OPerm (2)



H[i] is given, P[i] is new. Exploit the properties of the chaining bc:

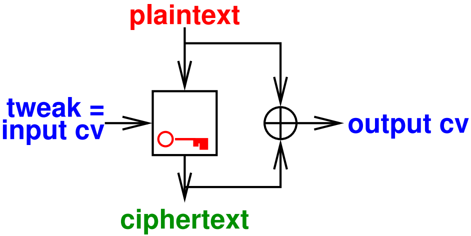
- **Good block cipher:** C[i] is like a random value.
- **Good keyed hash function:** H[i+1] has never been used before as an **input cv**.
- **Good block cipher:** C[i+1] is like a random value.
- **Good keyed hash function:** H[i+2] has never been used before as an **input cv**.
- ...

What if the last plaintext block P[m] is not a full block?

- Ciphertext stealing does not work.
- New approach: "**tag splitting**". See the paper.

4. Implementation (Chaining Block Cipher)

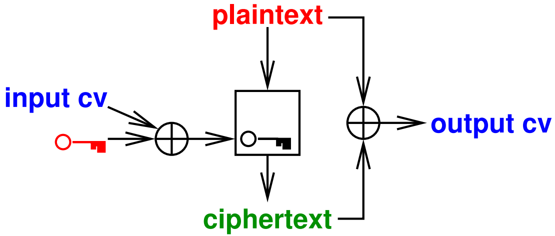
Use a tweakable block cipher, instead:



1. Set **tweak := input cv**
2. Set **output cv** := **plaintext** $\oplus$ **ciphertext**.

McOE-X - we don't have a Tweakable BC!

... at least no n-bit bc with n-bit tweaks – so use an ordinary one:

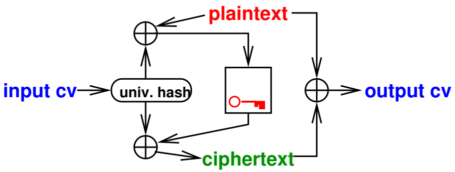


1. Xor the **input cv** into the **key**.
2. Set **output cv** := **plaintext** $\oplus$ **ciphertext** (as before).

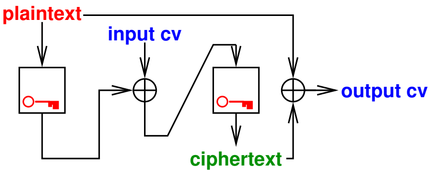
- Exposes the underlying block cipher to related-key attacks.
- Performs poor if the key schedule is slow.

Other Constuctions for a Chaining BC

McOE-G



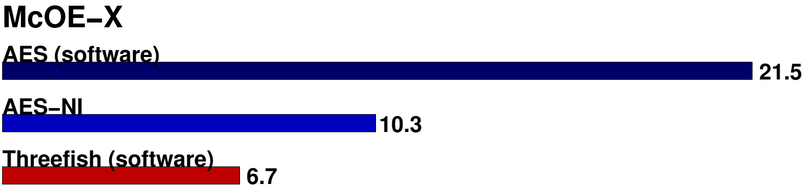
McOE-D



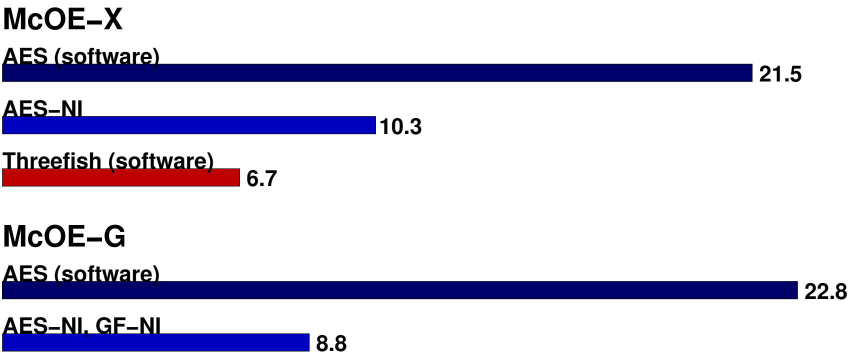
McOE-G: uses universal hash function H with Galois-Field arithmetic

McOE-D: uses double encryption

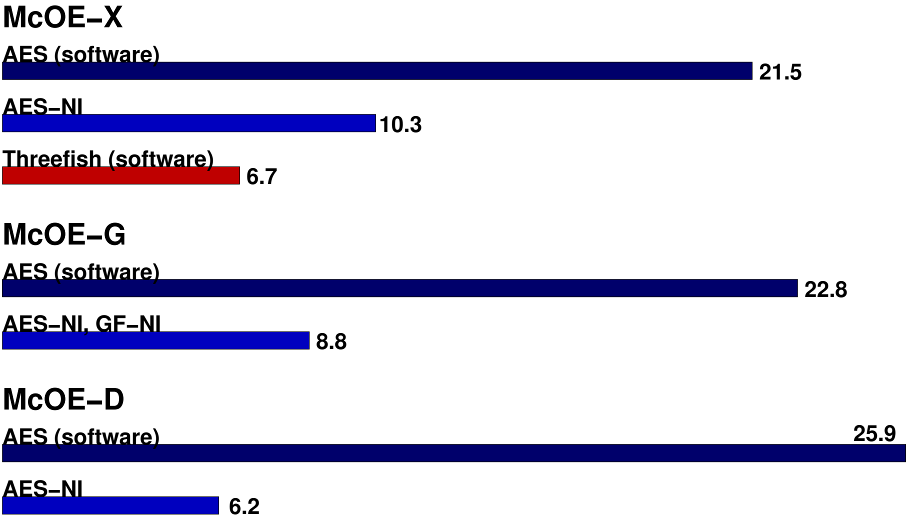
Throughput Values [cycles/byte]



Throughput Values [cycles/byte]



Throughput Values [cycles/byte]



5. Final Remarks

- If you are searching for new challenges regarding the **design of symmetric primitives**, we have one:
  - ⇒ Design efficient tweakable n-bit block ciphers with n-bit tweaks or highly key-agile ordinary block ciphers!

- **"This is not our problem":** Crypto applications fail because a cryptosystem is mistakenly used outside/against its specification.
- But when the same mistake is made again and again, then **maybe it is our problem** – and we should accept the challenge to **design misuse resistant cryptosystems!**

  Note that there are other misuse cases, beyond nonce reuse.