PB173 - Tématický vývoj aplikací v C/C++ (podzim 2012)

Skupina: Aplikovaná kryptografie a bezpečné programování

https://minotaur.fi.muni.cz:8443/pb173_crypto

Petr Švenda, *svenda@fi.muni.cz*
Konzultace: G201, Pondělí 16-16:50

Designing good API, authenticated encryption

Principles of good API

1. Be minimal
2. Be complete
3. Have clear and simple semantics
4. Be intuitive
5. Be easy to memorize
6. Lead to readable code

- read more at e.g., *http://doc.trolltech.com/qq/qq13-apis.html*
- security API even harder: *http://www.cl.cam.ac.uk/~rja14/Papers/SEv2-c18.pdf*
- *http://blog.apigee.com/taglist/security*

Read more about this topics

- Schneier on Security: http://www.schneier.com/
- TaoSecurity http://taosecurity.blogspot.com/
- Krebs on Security: http://krebsonsecurity.com/
- Freedom to Tinker: https://freedom-to-tinker.com/
- Light Blue Touchpaper: http://www.lightbluetouchpaper.org/
- …

Copy-free functions

- API style which minimizes array copy operations
- Frequently used in cryptography
  - we take block, process it and put back
  - can take place inside original memory array
- `int encrypt(byte array[], int startOffset, int length);`
  - encrypt data from *startOffset* to *startOffset* + *length;*
- Wrong(?) example:
  - `int encrypt(byte array[], int length, byte outArray[], int* pOutLength);`
  - note: C/C++ can still use pointers arithmetic
  - note: Java can't (we need to create new array)

Block cipher modes for Authenticated Encryption

Modes for authenticated encryption

- Encryption preserves confidentiality but not integrity
- Common integrity functions (like CRC) protect against random faults
- Cryptographic message integrity protects intensional errors

Confidentiality, integrity, privacy

- Message confidentiality [encryption]
  - attacker is not able to obtain info about plaintext
- Message integrity [MAC]
  - attacker is not able to modify message without being detected (PTX, CTX)
- Message privacy [encryption]
  - attacker is not able to distinguish between encrypted message and random string
  - same message is encrypted each time differently

Encryption and MAC composition

- Modes for block ciphers (CBC, CTR, CBC-MAC)
- Compositions (encryption + MAC)
  - encrypt-and-mac $[E_{Ke,Km}(M) = E_{Ke}(M) \mid T_{Km}(M)]$
    - can fail with privacy and authenticity
  - mac-then-encrypt $[E_{Ke,Km}(M) = E_{Ke}(M \mid T_{Km}(M))]$
    - can fail with authenticity
  - encrypt-then-mac $[E_{Ke,Km}(M) = E_{Ke}(M) \mid\mid T_{Km}(E_{Ke}(M)]$
    - always provides privacy and authenticity
- Paralelizability issue
- Authenticated-encryption modes (AE)
  - special block cipher modes for composed process

Usage scenarios

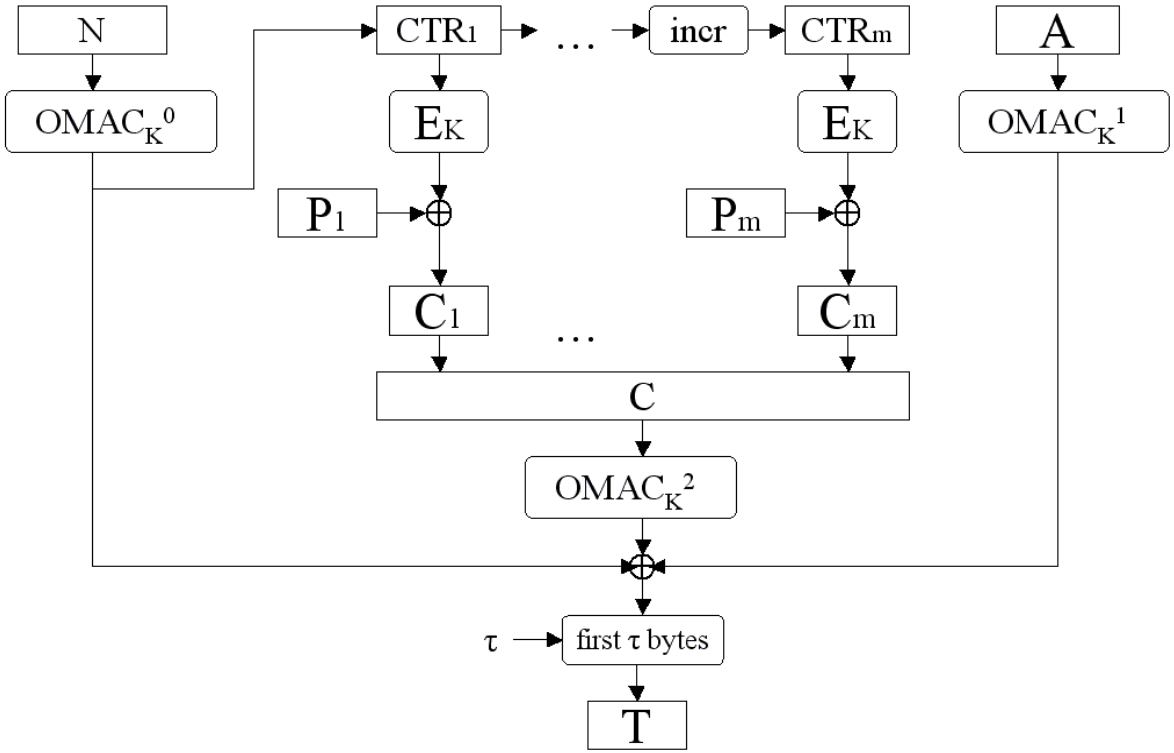
- Powerful, parallelizable environments
  - hardware accelerators
- Powerful, but almost serial environments
  - personal computer, PDA
- Restricted environments
  - smart card, cellular phone
- Different scenarios have different needs

Important features for AE modes

- Provable security
- Performance, paralelizability, memory req.
  - important for high-speed encryption, SC
- Patent
  - early AE modes were patented
- Associated data authentication
  - authentication of non-encrypted part
- Online, incremental MAC, number of keys, endian dependency …
- http://blog.cryptographyengineering.com/2012/05/how-to-choose-authenticated-encryption.html
- www.fi.muni.cz/~xsvenda/docs/AE_comparison_ipics04.pdf

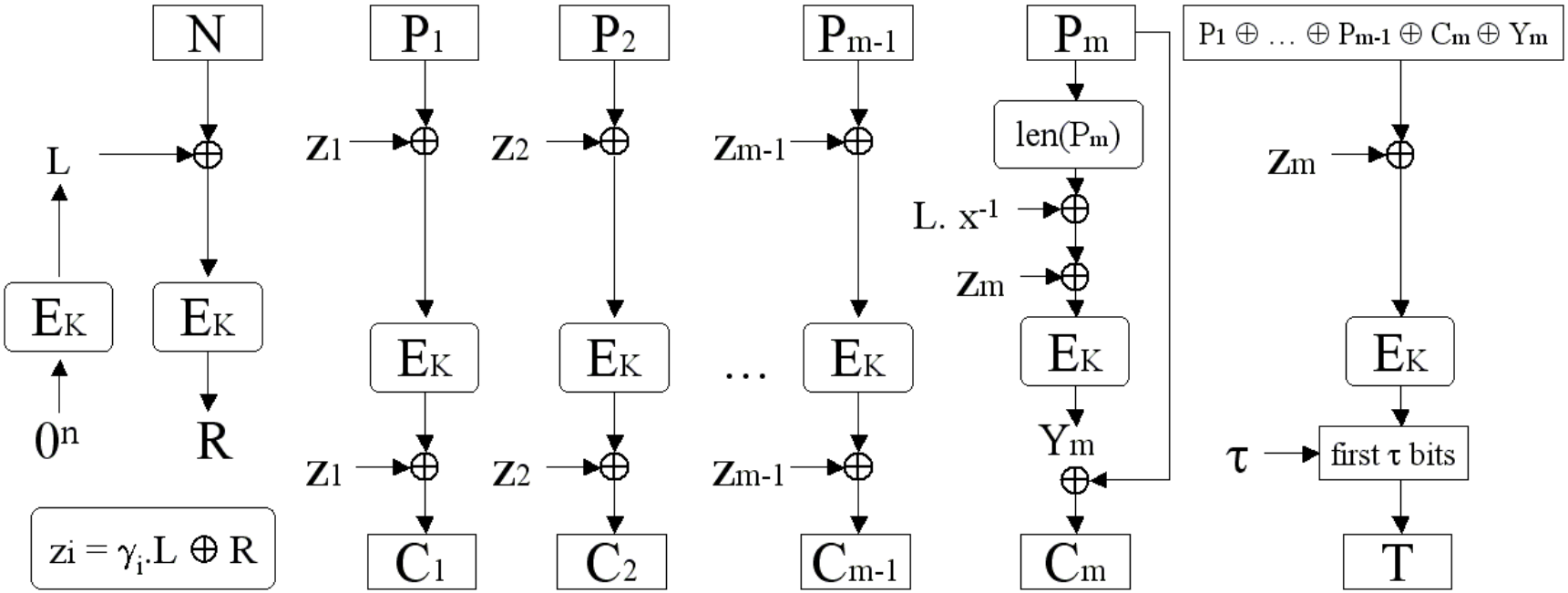
EAX mode

- Encrypt-than-mac composition
- Provable secure, unpatented



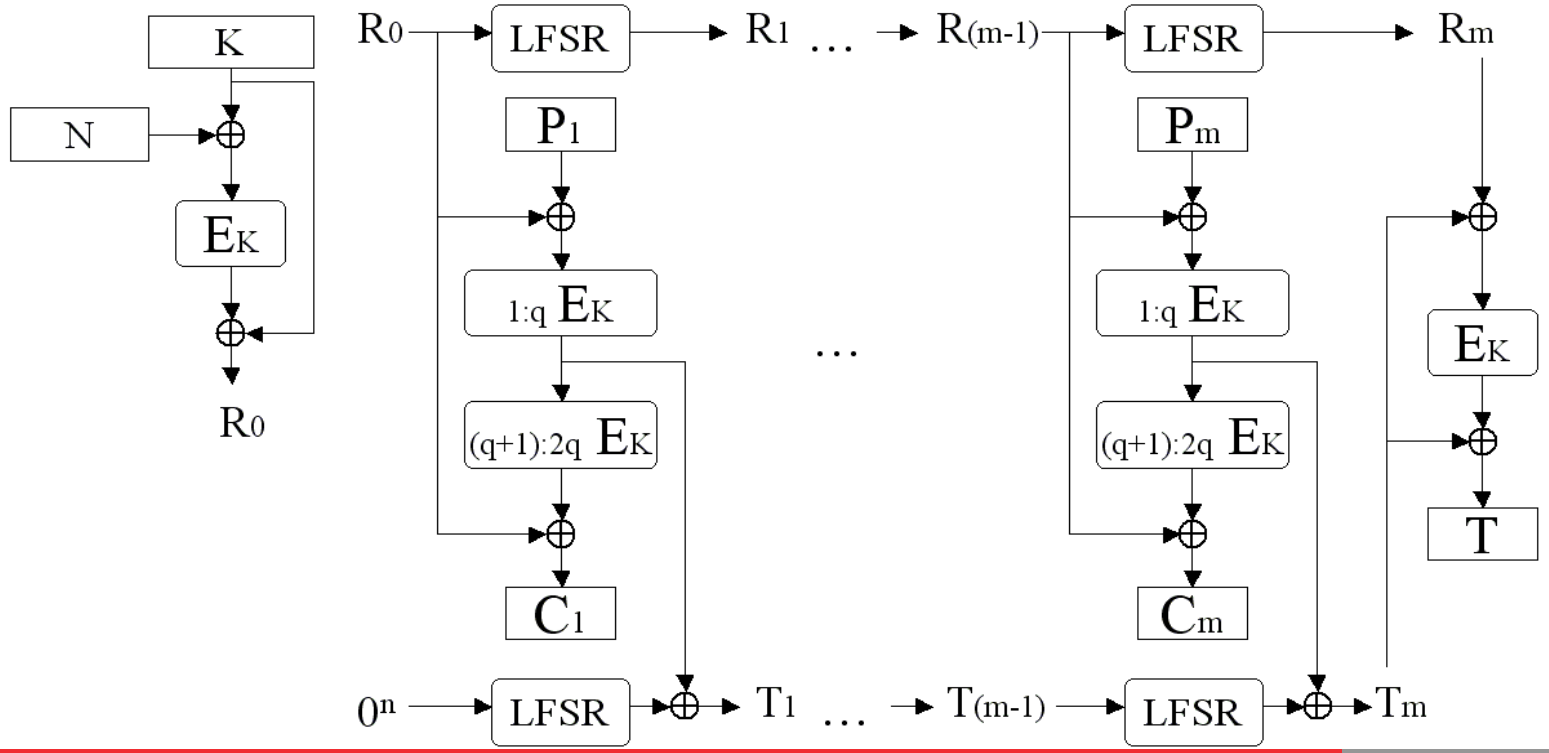
Offset CodeBook mode (OCB)

- Memory efficient, fast mode
- Provable secure, but patented



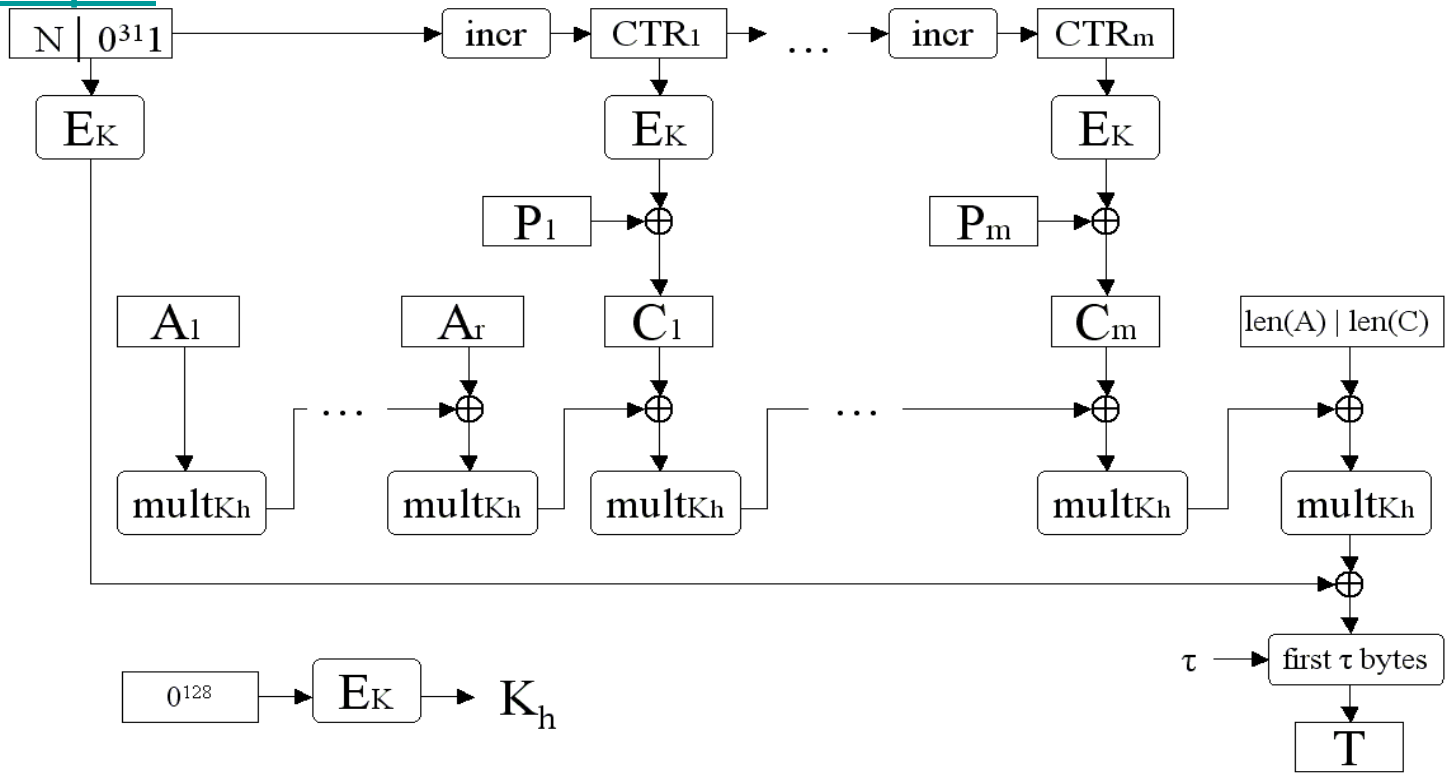
Cipher-State mode (CS)

- Memory efficient, fast mode, unpatented
- Not provable secure (inner state of cipher)



Galois/Counter Mode (GCM)

- Need pre-computed table (4kB-64kB)
- fast mode, provable secure, unpatented, NIST standard
- http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf



Implementation: AES-GCM from PolarSSL

- gcm.h, gcm.c

```
int gcm_init( gcm_context *ctx,
      const unsigned char *key,
      unsigned int keysize );

int gcm_crypt_and_tag( gcm_context *ctx,
      int mode, // GCM_ENCRYPT (alternatively GCM_DECRYPT)
      size_t length,
      const unsigned char *iv,
      size_t iv_len,
      const unsigned char *add, // authenticated, but not encrypted
      size_t add_len,
      const unsigned char *input, // authenticated and encrypted
      unsigned char *output,      // encrypted data
      size_t tag_len,
      unsigned char *tag );
```

```
int gcm_auth_decrypt( gcm_context *ctx,
      size_t length,              // length of input data
      const unsigned char *iv,
      size_t iv_len,
      const unsigned char *add,   // authenticated, but not encrypted
      size_t add_len,
      const unsigned char *tag,   // authenticator (MAC value)
      size_t tag_len,
      const unsigned char *input, // encrypted data
      unsigned char *output );    // decrypted data
```

Conclusions

- Composition of ENC and MAC can fail
  - encrypt-then-mac provable secure
  - specially designed composed modes
- Most promising mode is patented (OCB)
  - fast alternative GCM, CS
- Suitable mode depends on usage
  - paralelizability, memory
  - specific needs (online, incremental MAC)

Practical assignment

- Design, document and implement API to:
  1. **prepare protected content** (should be already done)
  2. **obtain data from protected content** (should be already done)
  3. **secure license**
     - design own xml format (limited to particular user, max 10 plays)
     - TinyXML, RapidXML… parser
     - add encryption and integrity protection
  4. **verify security of license** (add decryption and verify signatures)
- Write unit tests for all this tasks
- Document functions in JavaDoc-style (Doxygen)
  - and generate documentation with function call graph