Applied Cryptography
CPEG 472/672
Lecture 7A

Instructor: Nektarios Tsoutsos

Keyed Hashing

- Anyone can compute a hash value
  - There is no secret involved in the process
  - Attackers can modify a msg and create hash
- Is there a way to prevent that?
  - We want a type of hash for authorized users
  - Only those who know a secret can generate the hash, or can verify the hash
- Keyed Hashing
  - Message Authentication Codes (MACs)
  - Pseudo Random Functions (PRFs)

MACs

- Validate if a message has been modified
  - Protect msg integrity and authenticity
  - Create a tag T=MAC (key, msg)
  - Return pair (M, T)
- Can generate/verify tag iff you know K
  - Attackers do not know K
- Uses of MACs
  - Network packets IPSec, SSH, TLS
  - 3G/4G mobile telephony standards

Secure MACs

Define security goals/attack models

- Forgery
  - Create valid (msg, tag) pair without K
  - In secure MACs this should be impossible
  - Security goal: Unforgeability
- Attack Model
  - Basic model: known-msg attack (KMA)
    - Attacker collects valid (M,T) pairs
  - Standard model: chosen-msg attack (CMA)
    - Attacker asks oracle for tags of chosen msgs
- Replay attack
  - Attacker sends duplicate valid (M,T) pair
  - We want replay resistance (unique msg IDs)

PRFs & PRF security

- Inputs: message M, secret key K
- Output: indistinguishable from random
  - Attack cannot tell if a value is the output of a PRF or truly random without knowing K
  - Cannot find patterns for distinguishing PRF output from truly random values
- Notion: indistinguishability from random
- Used in other crypto constructions
  - Feistel networks, challenge/response ID
    - Server sends chal msg M, client sends PRF(K,M)
  - PRP/PRF Switching lemma (https://eprint.iacr.org/2004/331.pdf)

MACs vs PRFs

- Both keyed hashes but PRFs are stronger
  - MACs: weaker security requirements
    - Are secure if cannot be forged
    - MAC outputs cannot be guessed
  - PRFs: outputs IND-able from random strings
    - Stronger requirements compared to MACs
    - If PRF output IND-able => can't be guessed
- Any secure PRF is also a secure MAC
  - But: secure MAC not necessarily secure PRF
    - PRF2(K,M)=sec_PRF(K,M) || 0
    - PRF2 is not a secure PRF but a secure MAC

Can distinguish PRF2 from random

If you can forge a tag for PRF2 you can forge a tag for sec_PRF, which should be impossible

Constructions from hashes

- Secret-prefix construction: Hash(K||M)
  - Vulnerable to length extension attack
    - Compute Hash(K||M_1||M_2) given Hash(K||M_1)
      - Forgery: you don't know K or M_1
    - SHA-3 finalists not vulnerable to LEA
      - Mandatory NIST requirement for SHA-3
  - Insecurity with variable key lengths
    - Consider: Hash (K||M) with K=123abc, M=def000
      - Same as K=123a, M=bcdef000
    - Solution: Encode key length as well
      - Hash(L||K||M) with L=encode_length(K)

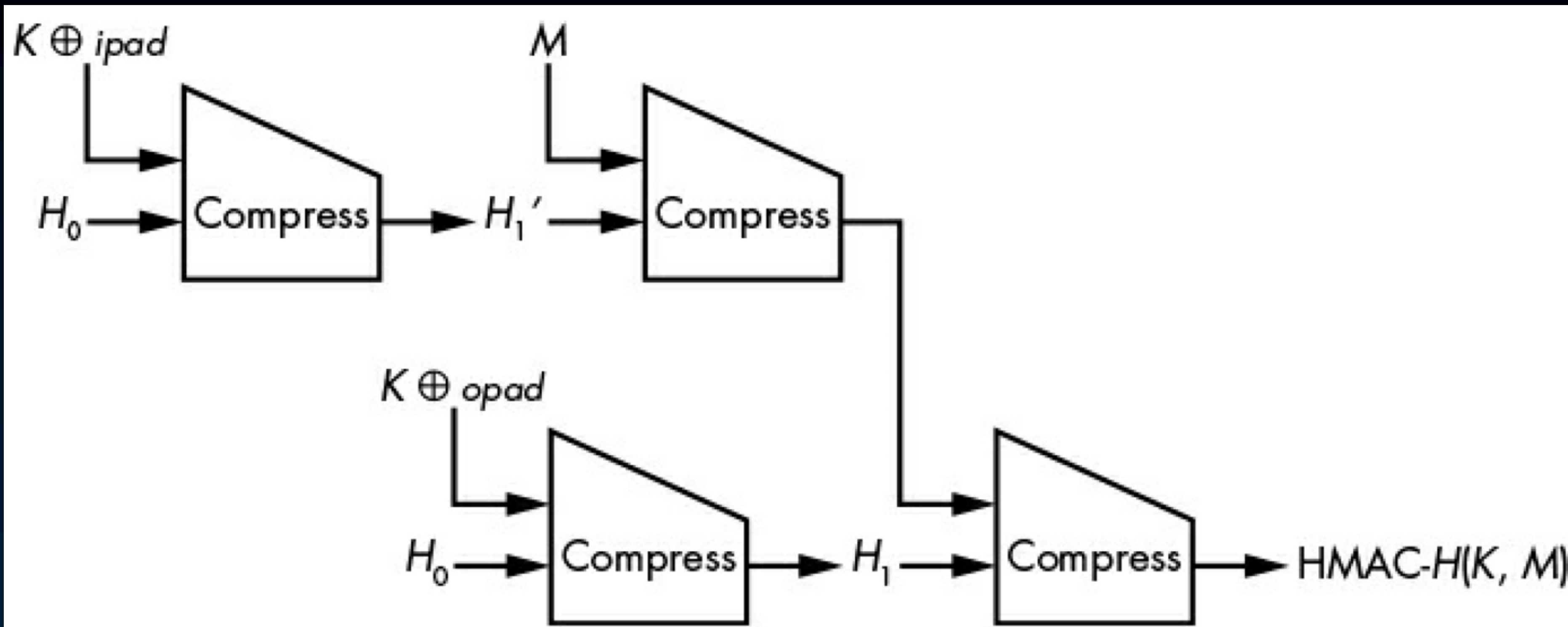
Constructions from hashes

- Secret-suffix construction: Hash(M||K)
  - Solves the LEA problem
    - Hash(M_1||K||M_2) is not extension of Hash(M_1||K)
      - The extension would be Hash(M_1|| M_2||K) instead
  - Vulnerable if Hash has collisions
    - Assume Hash(M_1)==Hash(M_2)
    - Do CMA attack: Request Hash(M_1||K)
    - Return pair M2, tag=Hash(M_2||K)
- Envelope method: Hash(K||M||K)
  - More secure than prefix/suffix constructions

Constructions from hashes: HMAC

- Secure PRF construction from a hash H
  - Needs collision resistant hash
    - At least PRF as compression function
  - Defines two paddings: opad, ipad
    - opad = 0x5c5c…5c5c (as long as blk size of H)
    - ipad = 0x3636…3636 (as long as blk size of H)
  - Pad K with 0x00’s to match blk size of H
- Definition of HMAC PRF using H
  - PRF=H((K xor opad) || H((K xor ipad)||M))
  - More secure than envelope construction

HMAC

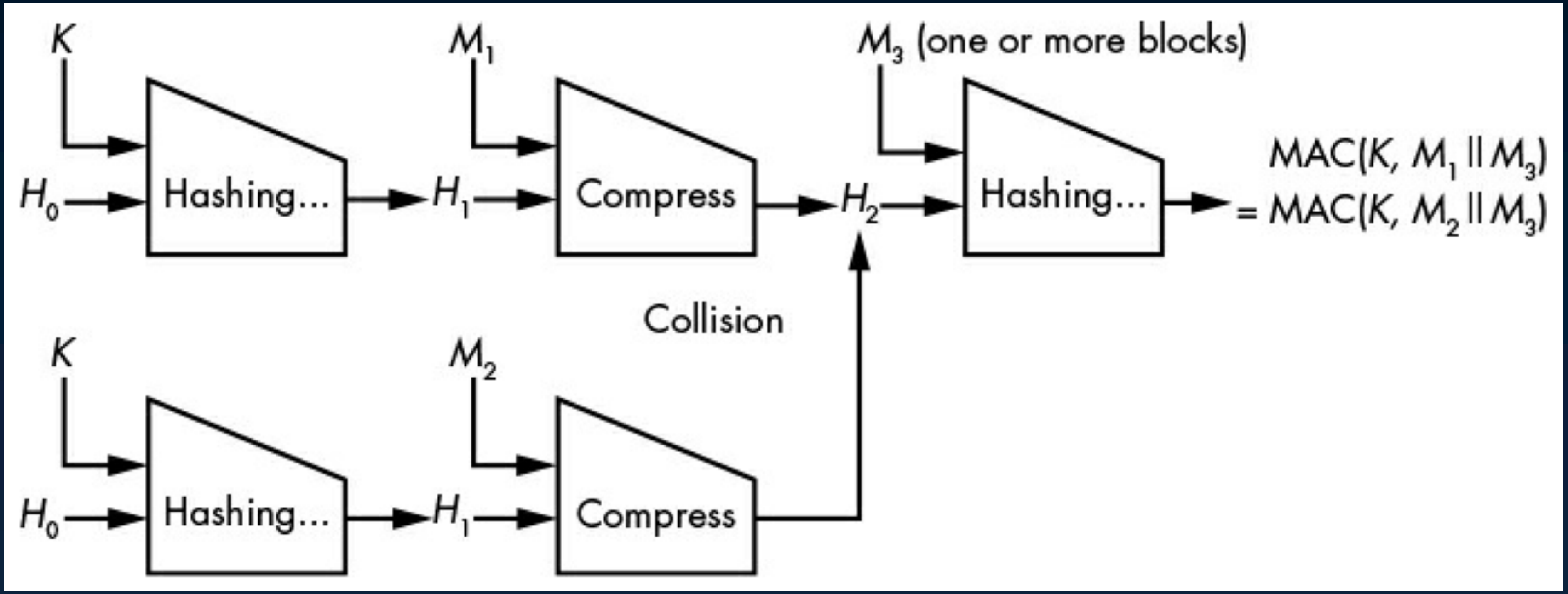


K ⊕ ipad
M
H_0
Compress
H_1'
Compress
K ⊕ opad
H_0
Compress
H_1
Compress
HMAC-H(K, M)

Generic attack on hash-based MACs

Applicable to MACs based on iterated hash functions

- Collision on hash => MAC collision
  - Assume Hash(K||M1)==Hash(K||M2)
  - Assume a LEA possible on Hash (e.g., SHA2)
  - Request 2^(n/2) tags (i.e., CMA attack)

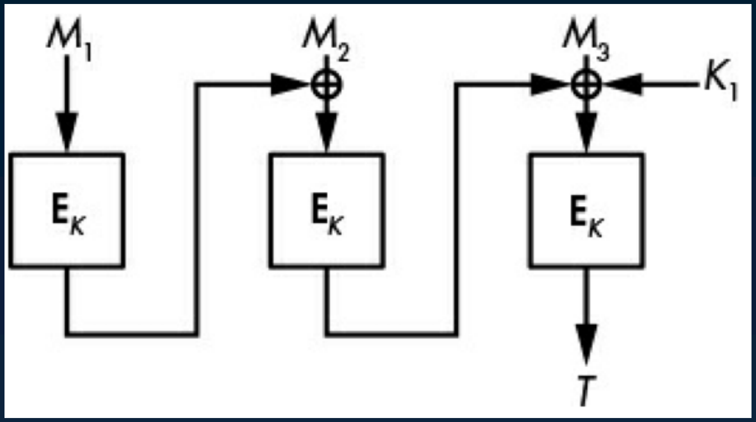


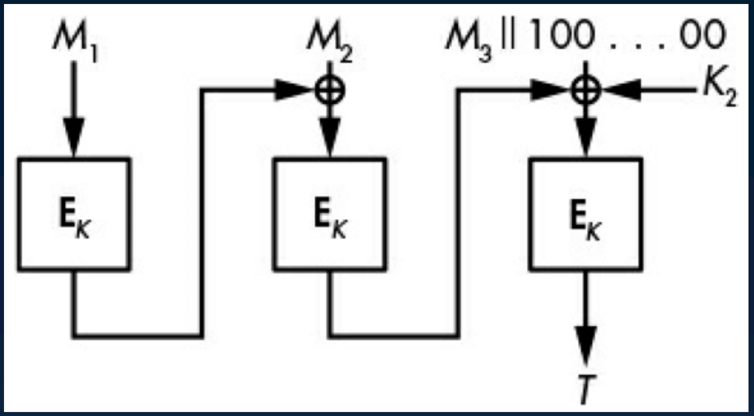
Keyed hashes from block ciphers

- Block ciphers already used before
  - E.g., HMAC-SHA-256 uses block cipher => D-M compression function => M-D hash
  - Can we use block cipher directly?
- CBC-MAC (broken)
  - Tag=Encrypt M with CBC, keep last ctxt blk
    - Keep C_i = E(K,M_i xor C_i-1) when i is last blk
    - Use 0x00..000 as IV
  - Forgery: given T1=E(K,M1), T2=E(K,M2) create new pair T2, (M1 || (M2 xor T1))

Keyed hashes from block ciphers

- Solution: CMAC
  - Fixes CBC-MAC using two extra keys K1, K2
    - L=E(0x00,K) encrypt K with zero as key
    - K1=L<<1 if MSB(L)=0 else (L<<1) xor 87
    - K2=K1<<1 if MSB(K1)=0 else (K1<<1) xor 87
  - Use K1 if M does not need padding, else K2





Dedicated MAC designs

Algorithms created to specifically serve as PRFs and MACs

- Universal hash functions (UHs)
  - Weaker notion vs cryptographic hashes
  - UHs don't need to be collision resistant
    - Only requirement: UH(K,M1)==UH(K,M2) with negligible probability
    - Use a secret key
  - No need to be pseudorandom like PRFs
- UHs can be used as ONE TIME MACs
  - Security collapses if used twice

Example UH using polynomials

- Select prime number p
- Select secret values R, K in range [1,p]
- Define UH(R,K,M) of message M as

  $$R+M1*K+\ldots+Mn*K\text{^}n \bmod p$$

- Limitations:
  - Can be used only for one msg M
  - Attacker can break after requesting 2 tags
    - Recover R by requesting UH(R,K,0)=R
    - Recover K by requesting UH(R,K,1)=R+K

Carter Wegman (C-W) construction

- Convert 1-time MAC to many-time MAC
  - "Encrypt" the UH using a PRF
  - MAC(K1,K2,N,M)=UH(K1,M)+PRF(K2,N)
    - Use nonce N (use only once per key K2)
    - PRF acts like a stream cipher
- Poly1305-AES: very fast C-W MAC
  - Faster than HMAC- or CMAC-based MACs
  - Poly1305(K1,M)+AES(K2,N) % 2^{128}
  - Define UH
    - Poly1305(K,M)=M_1*K^n+..+M_n*K % 2^{130}-5
    - Integer at most 129 bits

Timing attacks on MAC verification

- Problem: Variable time in equality check
  - Servers check the provided tag byte by byte
  - Use early termination if bytes are different

```
def compare_mac(x, y, n):
    for i in range(n):
        if x[i] != y[i]:
            return False
return True
```

- Information leakage: divide & conquer
  - Attacker can tell if tag byte was correct
  - Solution: constant-time implementation

Hands-on exercises

- Poly1305-AES
- HMAC
- CMAC
- SipHash

Reading for next week

- Aumasson: Chapter 8 (up to AES-GCM)
  - We will have a short quiz
- Midterm: Thu April 9th, 2:00-3:15pm
  - All material during first 6 weeks
  - Chapters 1-6
  - Lectures 1A-6B