CSE 484 (Winter 2008)

Applied Cryptography

Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Goals for Today

- Symmetric
- Reminder: Midterm on Friday. (Closed book.)
  - Contents up through the material for today
  - Not as hard as last year's midterm.
  - Make sure you understand the core concepts so far in this course:
    - Threat modeling
    - Software security
      - Problems
      - Defensive approaches
    - Symmetric cryptography
      - Components, definitions, security properties, classic problems

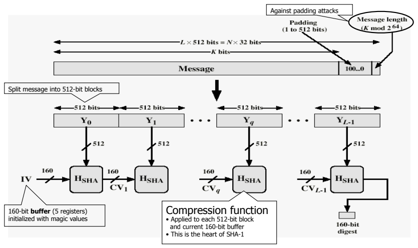
Which Property Do We Need?

- UNIX passwords stored as hash(password)
  - One-wayness: hard to recover password
- Integrity of software distribution
  - Weak collision resistance
  - But software images are not really random… maybe need full collision resistance
- Auction bidding
  - Alice wants to bid B, sends H(B), later reveals B
  - One-wayness: rival bidders should not recover B
  - Collision resistance: Alice should not be able to change her mind to bid B' such that H(B)=H(B')

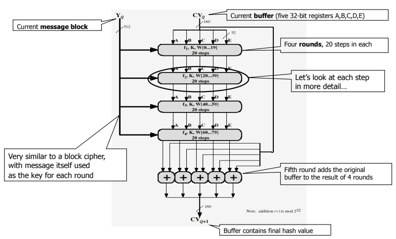
Common Hash Functions

- MD5
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- RIPEMD-160
  - 160-bit variant of MD5
- SHA-1 (Secure Hash Algorithm)
  - 160-bit output
  - US government (NIST) standard as of 1993-95
    - Also the hash algorithm for Digital Signature Standard (DSS)

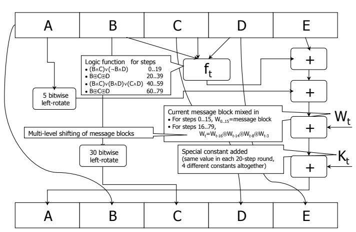
Basic Structure of SHA-1 (Skip)



SHA-1 Compression Function (Skip)



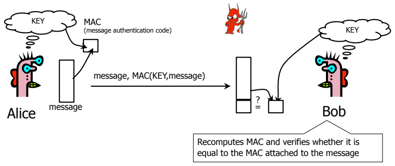
One Step of SHA-1 (80 steps total) (Skip)



How Strong Is SHA-1?

- Every bit of output depends on every bit of input
  - Very important property for collision-resistance
- Brute-force inversion requires 2^{160} ops, birthday attack on collision resistance requires 2^{80} ops
- Some very recent weaknesses (2005)
  - Collisions can be found in 2^{63} ops

Authentication Without Encryption

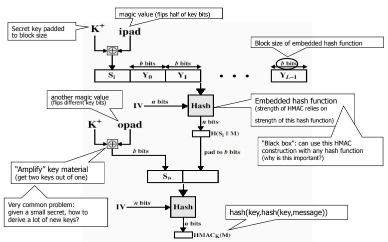


Integrity and authentication: only someone who knows KEY can compute MAC for a given message

HMAC

- Construct MAC by applying a cryptographic hash function to message and key
  - Could also use encryption instead of hashing, but…
  - Hashing is faster than encryption in software
  - Library code for hash functions widely available
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption
- Invented by Bellare, Canetti, and Krawczyk (1996)
  - HMAC strength established by cryptographic analysis
- Mandatory for IP security, also used in SSL/TLS

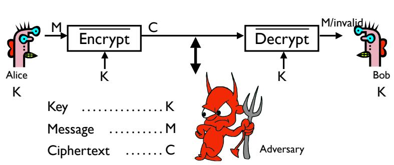
Structure of HMAC



Achieving Both Privacy and Integrity

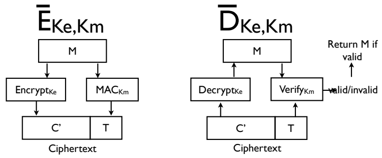
Authenticated encryption scheme

Recall: Often desire both privacy and integrity. (For SSH, SSL, IPsec, etc.)



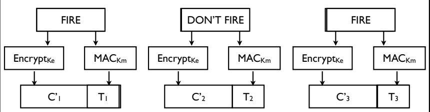
Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.



But insecure! [BN, Kra]

Assume Alice sends messages:



If $T_i = T_j$ then $M_i = M_j$

Adversary learns whether two plaintexts are equal.

Especially problematic when $M_1, M_2, \ldots$ take on only a small number of possible values.

The Secure Shell (SSH) protocol is designed to provide:

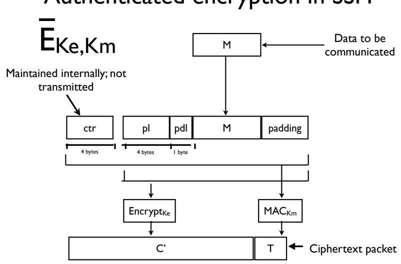
- Secure remote logins.
- Secure file transfers.

Where security includes:

- Protecting the privacy of users' data.
- Protecting the integrity of users' data.

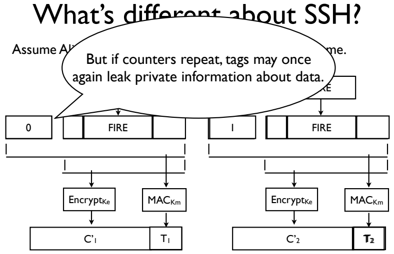
OpenSSH is included in the default installations of OS X and many Linux distributions.

Authenticated encryption in SSH



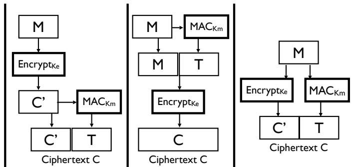
What's different about SSH?

Assume A… …me.



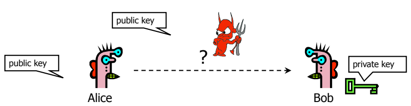
Then the tags T_1 and T_2 will be different with high probability.

Results of [BN00,Kra01]



	Encrypt-then-MAC	MAC-then-Encrypt	Encrypt-and-MAC
Privacy	Strong (CCA)	Weak (CPA)	Insecure
Integrity	Strong (CTXT)	Weak (PTXT)	Weak (PTXT)

Basic Problem



Given: Everybody knows Bob's public key
Only Bob knows the corresponding private key

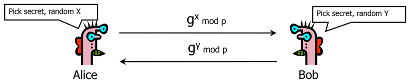
Goals: 1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate himself

Applications of Public-Key Crypto

- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric crypto, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (maybe)
    - Secret is stored only at one site: good for open environments
- Digital signatures for authentication
  - Can "sign" a message with your private key
- Session key establishment
  - Exchange messages to create a secret session key
  - Then switch to symmetric cryptography (why?)

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- <u>Public</u> info: p and g
  - p is a large prime number, g is a generator of Z_p^*
    - $Z_p^* = \{1, 2 \ldots p-1\}$; $\forall a \in Z_p^*\ \exists i$ such that $a = g^i \bmod p$
    - <u>Modular arithmetic</u>: numbers "wrap around" after they reach p



Compute $k=(g^y)^x=g^{xy} \bmod p$ Compute $k=(g^x)^y=g^{xy} \bmod p$

Why Is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:
  given $g^x \bmod p$, it's hard to extract x
  - There is no known <u>efficient</u> algorithm for doing this
  - This is <u>not</u> enough for Diffie-Hellman to be secure!
- Computational Diffie-Hellman (CDH) problem:
  given g^x and g^y, it's hard to compute $g^{xy} \bmod p$
  - … unless you know x or y, in which case it's easy
- Decisional Diffie-Hellman (DDH) problem:
  given g^x and g^y, it's hard to tell the difference between $g^{xy} \bmod p$ and $g^r \bmod p$ where r is random

Properties of Diffie-Hellman

- Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against <u>passive</u> attackers
  - Eavesdropper can't tell the difference between established key and a random value
  - Can use new key for symmetric cryptography
    - Approx. 1000 times faster than modular exponentiation
- Diffie-Hellman protocol (by itself) does not provide authentication